Incremental Verification of Component-Based Timed Systems

Jacques Julliand, Hassan Mountassir, Emilie Oudot

Incremental Verification of Component-Based Timed Systems

J. Julliand

LIFC - Laboratoire d'Informatique de l'Université de Franche-Comté
16, route de Gray
25030 Besançon Cedex, France
Ph:+33 (0)3 81 66 61 51, Fax:+33 (0)3 81 66 64 50
E-mail: jacques.julliand@lifc.univ-fcomte.fr

H. Mountassir

LIFC - Laboratoire d'Informatique de l'Université de Franche-Comté
16, route de Gray
25030 Besançon Cedex, France
Ph:+33 (0)3 81 66 66 65, Fax:+33 (0)3 81 66 64 50
E-mail: hassan.mountassir@lifc.univ-fcomte.fr

E. Oudot

LIFC - Laboratoire d'Informatique de l'Université de Franche-Comté
16, route de Gray
25030 Besançon Cedex, France
Ph:+33 (0)3 81 66 20 78, Fax:+33 (0)3 81 66 64 50
E-mail: emilie.oudot@lifc.univ-fcomte.fr

Abstract: We are interested in the incremental development, by integration of components, of component-based timed systems, and in particular, in the preservation of their properties during such a development process. We model timed components with timed automata. Their composition is achieved with the classic parallel composition operator for timed automata. The specifications of these timed systems are expressed with the timed linear logic MITL (Metric Interval Temporal Logic).
To guarantee the preservation of properties during an incremental development process, we propose to use τ-simulation relations, adapted for timed systems. First, we extend the classic notion of τ-simulation with timed aspects. As in the untimed case, this relation, called timed τ-simulation, preserves safety properties. To preserve more properties, in particular liveness ones, we present another relation, called divergence-sensitive and stability-respecting (DS) timed τ-simulation. This last relation preserves all MITL properties (and thus liveness ones), but also strong non-zenoness and deadlock-freedom. Moreover, as we put ourselves in a component-based framework, we study if the relations are appropriate to the use of the composition operator that we consider. For this purpose, we study if the relations are compatible with this operator, and if composability and compositionality hold. These three properties are a way to reduce the cost of the verification of the preservation, or even to get it for free. It results that the timed τ-simulation is appropriate with the classic operator since the properties hold without any assumption. However, this is not the case for the DS timed τ-simulation.
We implemented the algorithmic verification of the simulations in a tool called VESTA (Verification of Simulation for Timed Automata). The structure of the tool was inspired from the one of the OPEN-KRONOS tool. This allows, as additionnal feature, to connect the models considered in VESTA to the modules of the verification platform OPEN-CAESAR. We show the interest of our method by applying it on a case study, concerning a production cell example.

Keywords: Timed τ-simulation, component-based timed systems, incremental development, preservation of properties, MITL

Biographical notes:
J. Julliand is a full Professor in the Computer Science Laboratory LIFC at the University of Franche-Comté. He was the director of LIFC. Particular research interests include modeling, testing and verification of critical systems.
H. Mountassir is a Professor at the University of Franche-Comté. His general research interests are in verification of protocols, embedded systems and assembly of components.
E. Oudot received a PhD degree in 2006 from Franche-Comté University. Her research interests focus on verification of real time systems. Since then, after a postdoc position she is a software engineer at LIFC laboratory.

1 Motivations

Component-based modeling is a method which receives more and more attention. In particular, timed systems are often modeled this way. Instead of modeling an entire system in one go, it consists in decomposing the system into a set of sub-systems, called components, and to model each component independently. The complete model of the system is obtained by putting together all the components thanks to some parallel composition operator. We distinguish two main classes of properties which can be expressed to guarantee the correctness of such models: local properties and global properties. Local properties express requirements about the behaviour of a component (or subgroup of components) while global properties concern the behaviour of the whole system. Model-checking is a verification method which can be used to ensure that properties hold on the model of the system. For both kind of properties, the procedure consists in general in performing the verification on the complete model. However, model-checking is known to be difficult to apply on large-sized systems. Indeed, it suffers of the so-called state space explosion problem, which is accentuated in the case of timed systems, due to the presence of timing constraints.

Incremental development processes represent an alternative to circumvent this problem. The idea is to obtain the complete model of a system gradually, and, from a verification point of view, to check properties at each step of the development, where the model is small enough for the verification to be run to completion. The goal of this paper is to show how these incremental development processes can be exploited for component-based timed systems, and to study the impact in practice of the use of such methods, compared to classic verification. We distinguish two kinds of incremental development methods: integration of components and refinement. Given a set of components $C_1, \cdots, C_n$, integration of components consists in considering one component (or group of components), for instance C_1, and to check its properties in isolation before integrating it with other components. An essential property in this kind of development is composability, i.e., already established properties of C_1 must be preserved by the integration. Refinement is another kind of incremental development. The principle is first to establish an abstract model of the system and to progressively add details to it until getting to a model representing the complete system. Of course, this refinement process must not bring incoherences w.r.t. the abstract model. In particular, properties which hold on the abstract model must be preserved on the detailed version. For component-based systems, refinement consists in giving an abstract model for each component, and then adding details to each one. From a verification point of view, the goal is double: checking local properties of the components on their abstract model, and verifying global properties on the entire abstract model, obtained by the assembling of all abstract components. An important property is compositionality, meaning that if each detailed version of the components refines the abstract one, then the complete detailed model is a refinement of the complete abstract model.
When using such incremental methods, a major issue concerns the preservation of already checked properties. A way to ensure preservation is to compare the behaviour of the models, i.e., the model on which verification is performed and the model on which preservation must be ensured. This comparison must be done on some criteria, depending on the properties which must be preserved. Several equivalence relations or preorders have been defined to compare two systems: equivalence relations are generally used to test the "equality" between two systems modulo the relation, while preorders rather represent an implementation relation. In (Glabbeck 1990), twelve equivalence relations such as bisimulation, simulation or trace equivalences, and their associated preorder, are defined for (untimed) transition systems and are ordered according to a linear-branching time hierarchy. These relations are reconsidered in (Glabbeck 1993) by taking into account internal activity of the systems.

We are interested in the simulation preorder. Indeed, this kind of relation has already been used in the untimed case as a formalization of the refinement process to guarantee preservation of properties. For instance, (Bellegarde, Julliand & Kouchnarenko 2000) formalizes the refinement of (untimed) transition systems as a kind of τ-simulation which preserves LTL properties.
We present here two τ-simulation relations taking into account the timing constraints of the systems: a timed τ-simulation which preserves all safety properties, and a so-called divergence-sensitive and stability-respecting (DS) timed τ-simulation with the ability of preserving all properties which can be expressed with the linear timed logic MITL (Metric Interval Temporal Logic), strong non-zenoness and deadlock-freedom.
A way to show the usefulness of these relations for incremental development and their impact in practice is to examine if they preserve composability and compositionality. Given components A and B, and some composition operator $\|$, composability is ensured if A simulates $A\|B$. The direct consequence is that local properties of A are automatically preserved during its integration (the kind of properties preserved depends on the notion of simulation considered). Given components A, B, C and D, compositionality means that if A simulates B and C simulates D then $A\|C$ simulates $B\|D$. We study these properties of the simulations w.r.t. the classic composition operator for timed systems, which uses a composition paradigm *a la CSP* (Hoare 1985). We show that the timed τ-simulation is well-adapted to incremental development achieved with this operator, since composability and

compositionality are guaranteed for free. However, this is not the case for the DS timed τ-simulation. Thus, to guarantee the preservation when using this operator, the DS timed τ-simulation has to be checked algorithmically. We implemented this verification in a tool named VESTA (Verification of Simulation for Timed Automata). With this tool, we performed experiments to ensure that an algorithmic verification of the simulation (and thus of the preservation) does not advances to incremental development comparing to a direct verification of the properties. The results obtained are encouraging since they show that, even when the DS timed τ-simulation is checked to ensure preservation, incremental development can speed up verification and that models that are too large to be verified in a whole can be checked this way.

The structure of the paper is the following. First, in section 2, we recall some background on timed systems. Section 3 presents the τ-simulations we define for timed systems, and their preservation abilities. In section 4, we show the usefulness of the simulations for incremental development by studying composability and compositionality w.r.t. the classic parallel composition operator that we consider. Section 5 is dedicated to the verification in practice of the simulations, and to experiments. We present some related works in section 6. Finally, section 7 contains the conclusion and plans the future works.

2 Modeling timed systems and their properties

In this section, we review some basics concerning timed systems. First, we present the model we consider for timed systems, i.e., timed automata, and timed composition operators. We also present the logic MITL that we use to express properties of timed systems.

2.1 Timed Automata

Timed automata (TA) (Alur & Dill 1994) are amongst the most studied models for continuous-time systems. They are finite automata extended with real-valued variables called clocks, modeling the time elapsing. We consider as time domain the set of non-negative reals $\mathbb{R}^+$. Before considering TA, we recall some usual definitions about clocks. Then, we present the syntax and semantics of TA, and the symbolic representation of the state-space of a TA.

Clock valuations. Let X be a set of clocks. A clock valuation over X is a function $v : X \to \mathbb{R}^+$ mapping to each clock in X a value in $\mathbb{R}^+$. Let $\mathbf{0}$ denote the valuation assigning 0 to each clock in X.
Operations on valuations. Let v be a valuation over X and $t \in \mathbb{R}^+$, the valuation $v + t$ (respectively $v - t$) is obtained by adding (resp. substracting) t to the value of each clock. Given $Y \subseteq X$, the dimension-restricting projection of v over Y, written $v\rfloor_Y$ is a new valuation over Y containing only the values in v of clocks in Y. The reset in v of the clocks in Y, written $[Y := 0]v$ is the valuation obtained from v by setting to zero all clocks in Y, and leaving the values of other clocks ($\in X \backslash Y$) unchanged.

Clock constraints and polyhedra. The set $\mathcal{C}_{df}(X)$ of diagonal-free clock constraints over X is defined by the following grammar:

$$g ::= x \sim c \mid g \wedge g \mid \mathit{true} \text{ where } x \in X,\ c \in \mathbb{N}, \text{ and } \sim \in \{<, \leq, =, \geq, >\}.$$

Diagonal-free constraints do not allow comparisons between clocks, of the form $x - y \sim c$. A valuation v over X satisfies a constraint $x \sim c$, written $v \in x \sim c$, if $v(x) \sim c$. The satisfaction of other constraints is defined as usual. Note that a clock constraint over X defines a convex X-polyhedron. Let `zero` denote the X-polyhedron defined by $\bigwedge_{x \in X} x = 0$.
Operations on polyhedra. The dimension-restricting projection and reset operations defined on valuations can be directly extended to polyhedra. The *backward diagonal projection* of the X-polyhedron ζ defines an X-polyhedron $\swarrow\zeta$ such that $v' \in \swarrow\zeta$ if $\exists \delta \in \mathbb{R}^+ \cdot v' + \delta \in \zeta$. Similarly, the *forward diagonal projection* of ζ defines an X-polyhedron $\nearrow\zeta$ such that $v' \in \nearrow\zeta$ if $\exists \delta \in \mathbb{R}^+ \cdot v' - \delta \in \zeta$. Given $c \in \mathbb{N}$, the extrapolation of ζ w.r.t c, written $\texttt{Approx}_c(\zeta)$, is the smallest polyhedron $\zeta' \supseteq \zeta$ defined intuitively as follows: lower bounds of ζ greater than c are replaced by c, and upper bounds greater than c are ignored. All these operations preserve the convexity of polyhedra. This property allows the simulation graph (see Def. 3) on a finite set of polyhedra and the infinite semantic graph (see Def. 2) of a timed automaton to be bisimilar.

Definition 1 (Timed Automaton) *Let* Props *be a set of atomic propositions. A timed automaton is a tuple* $A = \langle \mathrm{Q}, \mathrm{q}_0, \Sigma, \mathrm{X}, \mathrm{T}, \texttt{Invar}, \mathrm{L} \rangle$ *where:*

- Q *is a finite set of locations.*
- $q_0 \in Q$ *is the initial location of the automaton.*
- Σ *is a finite alphabet.*
- X *is a finite set of clocks.*
- $T \subseteq Q \times \mathcal{C}_{df}(X) \times \Sigma \times 2^X \times Q$ *is a finite set of edges.*
- $\texttt{Invar} : Q \to \mathcal{C}_{df}(X)$ *is a function associating a time-progress condition (called invariant) to each location.*
- $L : Q \to 2^{\text{Props}}$ *is the labelling function mapping a set of atomic propositions to each location.*

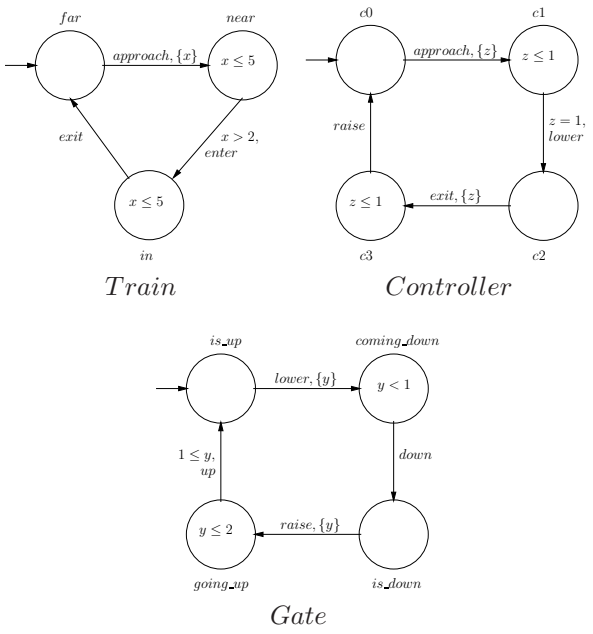

Figure 1 Timed automata of the train, the controller and the gate

An edge is written as a tuple $e = (q, g, a, r, q')$ where q and q' are the source and target locations, g is a clock constraint defining the guard of the edge, a is the label of the edge and r is the set of clocks to be reset by the edge. In the sequel, we use the notations $\texttt{label}(e)$ and $\texttt{reset}(e)$ to denote respectively a and r.

Example 1 *As a running example, we use the well-known railroad crossing taken from (Alur 1991). It is made up of at least three elements: one or several trains, a gate and the controller of the gate. Each element is subject to strong timing constraints. The expected global behaviour of this system is the following. When a train arrives near the crossing, it sends a signal to the controller to inform of its approach. One time unit (t.u.) later, the controller commands the closing of the gate which must be down within the following t.u. The train enters the crossing, and its passage lasts at most three t.u. When it exits the crossing, it sends a signal to the controller which commands the opening of the gate during the following t.u. The gate must respond and be up within the next t.u. Fig. 1 shows the timed automata modeling each component of this system. Each location is designated by a name (the label associated by the function L) and a clock constraint representing its invariant. For more readability, guards and invariants equal to true are omitted, as well as empty resets on edges.*

Definition 2 (Semantic Graph) *The semantics of a TA* $A = \langle \mathrm{Q}, \mathrm{q}_0, \Sigma, \mathrm{X}, \mathrm{T}, \texttt{Invar}, \mathrm{L} \rangle$ *is an infinite graph where states are pairs* (q, v), $q \in Q$ *and* v *is a clock valuation over* X *such that* $v \in \texttt{Invar}(q)$. *Its initial state is the pair* $(q_0, \mathbf{0})$. *For a state* $s = (q, v)$, *we call* $\texttt{disc}(s)$ *the discrete part* q *of* s. *The transitions of this graph can be either discrete transitions or time transitions:*

- *Discrete transitions: given an edge* $e = (q, g, a, r, q')$ *of* A, $(q, v) \xrightarrow{g,a,r} (q', v')$ *is a discrete transition in the semantics of* A *if* $v \in g$. *The valuation* $v' \in \texttt{Invar}(q')$ *is obtained by resetting in* v *all clocks in* r. *We call* (q', v') *a discrete successor of* (q, v). *We also directly write* $(q, v) \xrightarrow{e} (q', v')$ *such a transition, or simply* $(q, v) \xrightarrow{a} (q', v')$ *when the other elements are irrelevant.*
- *Time transitions have the form* $(q, v) \xrightarrow{t} (q, v + t)$ *where* $t \in \mathbb{R}^+$ *and* $v + t \in \texttt{Invar}(q)$. *We say that* $(q, v + t)$ *is a time successor of* (q, v). *Given a state* $s = (q, v)$, *we also use the notation* $s + t$ *for the pair* $(q, v + t)$.

In the sequel, we directly say states and transitions of the TA A, instead of states and transitions of the semantic graph of A.

Runs. A run of a TA A is a path of its semantic graph. Thus, a run is a finite or infinite sequence $\rho = (q_0, v_0) \xrightarrow{t_0} (q_0, v_1) \xrightarrow{e_0} (q_1, v_2) \xrightarrow{t_1} (q_1, v_3) \xrightarrow{t_2} (q_1, v_4) \xrightarrow{e_1} (q_2, v_5) \cdots$.
Note that we do not concatenate successive time transitions in a run. In the rest of the paper, we note (ρ, k) the k^{th} state of ρ and $\Gamma(A)$ represents the set of runs of A. With the definition of runs, we can now define what a reachable state is. A state (q_i, v_i) of A is said *reachable* if there exists some run of A visiting it, i.e. $\exists \rho \cdot (\rho \in \Gamma(A) \wedge \rho = (q_0, v_0) \xrightarrow{t_0} (q_0, v_1) \xrightarrow{e_0} (q_1, v_2) \xrightarrow{t_1} \cdots (q_i, v_i) \cdots)$.
Non-zenoness. A run is called non-zeno if time can diverge along the run. We write $\texttt{time}(\rho, k)$ to denote the time elapsed from the initial state of the run ρ until its k^{th} state, $\texttt{time}(\rho, s)$ for the time elapsed from the initial state of ρ until the state s and $\texttt{time}(\rho)$ for the total time elapsed in the run. Given a run ρ, if $\texttt{time}(\rho) = \infty$, then ρ is non-zeno (Tripakis 1998). A TA is said strongly non-zeno if all its runs are non-zeno. A weakest notion of non-zenoness can also be considered, expressing that there exists no reachable state which is zeno, i.e., such that all runs leaving from it are zeno.

Remark 1 (Timed state sequences) *The executions of a TA can also be expressed in terms of timed state sequences (TSS) instead of runs. A TSS is a sequence* $\sigma = (q_0, I_0) \xrightarrow{e_0} (q_1, I_1) \xrightarrow{e_1} \cdots$ *alternating pairs and discrete transitions. The* q_i *and* e_i *are respectively locations and edges of the TA, while the* I_i *are closed intervals representing the time elapsing before some discrete transition is taken. We say that a run* $\rho = (q_1, v_1) \xrightarrow{t_1} (q_1, v_1') \xrightarrow{e_1} (q_2, v_2) \xrightarrow{t_2} (q_2, v_2') \xrightarrow{t_2'} (q_2, v_2'') \xrightarrow{e_2} \cdots$ *is inscribed in a TSS* $\sigma = (q_1, I_1) \xrightarrow{e_1} (q_2, I_2) \xrightarrow{e_2} \cdots$ *if* $\forall i = 1, 2, ..., \texttt{time}(\rho, (q_i, \_)) \in I_i$.
Note that a run is inscribed in a unique TSS, and that there exists an infinite number of runs inscribed in a single TSS, since successive time transitions are not

concatenated. We use the notation $\sigma(\rho)$ for the TSS in which is inscribed the run ρ and (σ, i) the i^{th} pair of the TSS σ. Given $t \in I_i$, we write σ^t the suffix of a TSS σ at time t, where $\sigma^t = (q_i, I_i - t) \xrightarrow{e_i} (q_{i+1}, I_{i+1} - t) \xrightarrow{e_{i+1}} (q_{i+2}, I_{i+2} - t) \cdots$.

Symbolic representation. The semantic graph of a TA has an infinite number of states. To get a finite representation of this graph, the symbolic representation currently used is based upon the notion of zones, and leads to a symbolic graph called *simulation graph*.

Zones. A zone is a symbolic state which groups together states of a TA A such that they have the same discrete part, and the set of their valuations forms a convex polyhedron. Thus, a zone z is a pair (q, ζ) where q is a location of A and ζ is a convex polyhedron. We note $\texttt{disc}(z)$ the discrete part q of the zone z, and $\texttt{poly}(z)$ its polyhedron.

Operations on zones. The operations $\texttt{time-succ}(z)$ and $\texttt{time-pred}(z)$ define respectively the set of time successors and predecessors of some state in z. The operations $\texttt{disc-succ}(e, z)$ and $\texttt{disc-pred}(e, z)$ represent respectively the set of discrete sucessors and predecessors of some state in z by taking a discrete transition stemming from an edge e.

$$\begin{aligned}
\texttt{time-succ}(z) &\stackrel{def}{=} \{s' \mid \exists s \in z, t \in \mathbb{R}^+ \; s \xrightarrow{t} s'\} \\
\texttt{time-pred}(z) &\stackrel{def}{=} \{s \mid \exists s' \in z, t \in \mathbb{R}^+ \cdot s \xrightarrow{t} s'\} \\
\texttt{disc-succ}(e, z) &\stackrel{def}{=} \{s' \mid \exists s \in z \cdot s \xrightarrow{e} s'\} \\
\texttt{disc-pred}(e, z) &\stackrel{def}{=} \{s \mid \exists s' \in z \cdot s \xrightarrow{e} s'\}
\end{aligned}$$

Let us now define the successor and predecessor operations for zones. The operation $\texttt{post}(e, z, c)$ defines the successor zone of z by the transition e (w.r.t. a constant c), i.e., the set of states which can be reached from some states in z by taking transition e and letting time elapse. Note that the operator $\texttt{Approx}_c$ is also used in the definition of `post`, to ensure the termination of the construction of the simulation graph described below and which is based on this notion of zone. For more readability, in the definition of `post`, $\texttt{Approx}_c$ is applied on a zone instead of the polyhedron of the zone. The operation $\texttt{pre}(e, z)$ defines the predecessor zone of z by the discrete transition e, i.e., the set of states from which a state in z can be reached, by taking e and letting some time pass. Formally:

$$\begin{aligned}
\texttt{post}(e, z, c) &\stackrel{def}{=} \texttt{Approx}_c(\texttt{time-succ}(\texttt{disc-succ}(e, z))) \\
\texttt{pre}(e, z) &\stackrel{def}{=} \texttt{disc-pred}(e, \texttt{time-pred}(z))
\end{aligned}$$

Definition 3 (Simulation graph) *Let $A = \langle \mathrm{Q}, \mathrm{q}_0, \Sigma, \mathrm{X}, \mathrm{T}, \texttt{Invar}, \mathrm{L} \rangle$ be a timed automaton and $c \in \mathbb{N}$ a constant greater or equal to the greatest constant appearing in a constraint of A. The simulation graph of A w.r.t. c, written $SG(A, c)$, is a tuple $\langle \mathrm{Z}, \mathrm{z}_0, \Sigma, \mathcal{T} \rangle$ where:*

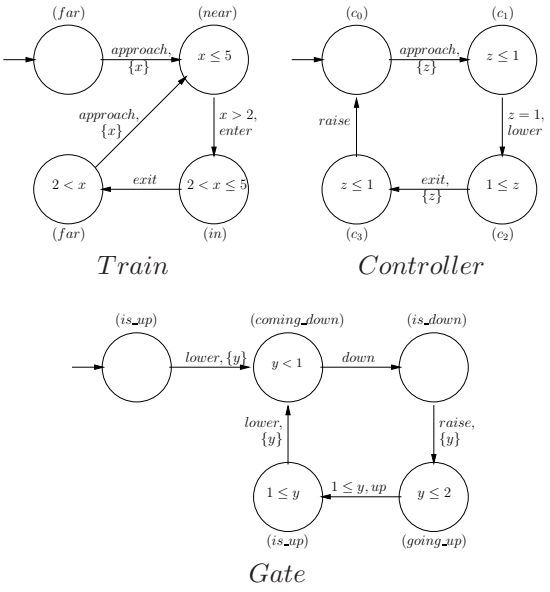

Figure 2 Simulation graphs of the train, the controller and the gate

- *Z is the finite set of states of the graph, which is a set of zones,*
- *$z_0 = (q_0, \nearrow \texttt{zero} \cap \texttt{Invar}(q_0))$ is the initial zone,*
- *$\mathcal{T} \subseteq Z \times T \times Z$ is the finite set of transitions. Given a zone z and an edge $e \in T$, if $z' = \texttt{post}(e, z, c) \neq \emptyset$, then z' is a zone of the graph and $z \xrightarrow{e} z'$ is a transition of the graph.*

Example 2 *Let us go back to the railroad crossing example as presented in Fig. 1. The simulation graphs built from each timed automaton are presented in Fig. 2. The polyhedron of each zone is represented (graphically) by a constraint given inside the zone.*

Paths and non-zeno paths in a simulation graph. A path in the simulation graph is a finite or infinite sequence $\pi = z_0 \xrightarrow{e_0} z_1 \xrightarrow{e_1} z_2 \cdots$. The set of paths of a simulation graph SG is written $\Pi(SG)$. A path is non-zeno if, for each clock $x \in X$, either x is reset infinitely often in the path, or x remains unbounded from one zone in the path. A formal definition can be found in (Tripakis 1998).

Relation between runs and paths. Each run (respectively non-zeno run) of A is inscribed in a unique path (resp. non-zeno path) of $SG(A, c)$, and for each path π of $SG(A, c)$ (resp. non-zeno path), there exists a run (resp. non-zeno run) inscribed in π (Tripakis 1998).

2.2 The classic composition operator for timed automata

We consider timed systems modeled in a compositional way. Each component is modeled as a TA. To put timed

components together, parallel composition operators which can handle timing informations have been defined. We focus on a particular kind of timed composition. We call it *classic parallel composition* since it is the classic composition used in the timed case.

This composition, written $\|$, operates between TA with disjoint sets of clocks. Intuitively, it is defined as a synchronized product where synchronizations are done on actions with identical label, while other actions interleave and time elapses synchronously between all the components.

Formally, let us consider two TA $A_i = \langle Q_i, q_{0_i}, \Sigma_i, X_i, T_i, \texttt{Invar}_i, L_i\rangle$ for $i = 1, 2$, such that $X_1 \cap X_2 = \emptyset$. The parallel composition of A_1 and A_2, written $A_1 \| A_2$, creates a new TA whose set of clocks is $X_1 \cup X_2$ and whose labels are in $\Sigma_1 \cup \Sigma_2$. The set Q of locations consists of pairs (q_1, q_2) where $q_1 \in Q_1$ and $q_2 \in Q_2$. The initial location is the pair (q_{0_1}, q_{0_2}). The invariant of a location (q_1, q_2) is $\texttt{Invar}(q_1) \wedge \texttt{Invar}(q_2)$, and its label is $L(q_1) \cup L(q_2)$. The set T of edges is defined by the following rules:

- Interleaving:

$$\frac{(q_1,q_2)\in Q \ , \ (q_1,g_1,a,r_1,q_1') \in T_1 \ , \ a\notin\Sigma_2}{((q_1,q_2),g_1,a,r_1,(q_1',q_2)) \in T}$$
$$\frac{(q_1,q_2)\in Q \ , \ (q_2,g_2,a,r_2,q_2') \in T_2 \ , \ a\notin\Sigma_1}{((q_1,q_2),g_2,a,r_2,(q_1,q_2')) \in T}$$

- Synchronization:

$$\frac{(q_1,q_2)\in Q,\ (q_1,g_1,a,r_1,q_1') \in T_1 \ , \ (q_2,g_2,a,r_2,q_2') \in T_2}{((q_1,q_2),g_1\wedge g_2,a,r_1\cup r_2,(q_1',q_2')) \in T}.$$

Example 3 *Let us go back to the railroad crossing example. This system is modeled by at least three components : one or several train, a gate and its controller. The parallel composition of the three timed automata representing respectively the train, the gate and the controller (as they are presented in Fig. 1) leads to the timed automaton given in Fig. 3.*

2.3 Metric Interval Temporal Logic

MITL (Metric Interval Temporal Logic) (Alur, Feder & Henzinger 1996) is a logical formalism allowing to express linear timed properties. It can be viewed as an extension of the linear (untimed) logic LTL (Linear Temporal Logic) (Pnueli 1981), where each temporal operator is constrained by a time interval. MITL formulas are defined inductively by the following grammar:

$$\varphi ::= ap \mid \neg\varphi \mid \phi \vee \psi \mid \phi \, \mathcal{U}_I \, \psi$$

where ap is an atomic proposition and I is a non-singular interval with integer bounds (a singular interval is of the form $[a, a]$, i.e., it is closed and its left and right bounds are equal). Other classic temporal operators can also be defined: $\Diamond_I \varphi = true \ \mathcal{U}_I \varphi$ (eventually φ within the interval I) and $\Box_I \varphi = \neg\Diamond_I \neg\varphi$ (always φ within the interval I).
MITL formulas are interpreted over timed state sequences. The satisfaction of a MITL formula φ over a timed state sequence σ, written $\sigma \models \varphi$, is defined as follows:

- $\sigma \models \texttt{true}$ is true,
- $\sigma \models ap$ iff $ap \in L(\texttt{disc}((\sigma, 0)))$,
- $\sigma \models \neg\varphi$ iff it is not true that $\sigma \models \varphi$,
- $\sigma \models \phi \vee \psi$ iff $\sigma \models \phi$ or $\sigma \models \psi$,
- $\sigma \models \phi \, \mathcal{U}_I \, \psi$ iff there exists $t \in I$ such that $\sigma^t \models \psi$, and $\forall t' \in (0, t)$, $\sigma^{t'} \models \phi$.

We say that a MITL formula φ is valid on a TA A, written $A \models \varphi$, iff φ is true over all the runs of A, i.e.:

$$A \models \varphi \text{ iff } \forall \rho \cdot (\rho \in \Gamma(A) \Rightarrow \sigma(\rho) \models \varphi).$$

Example 4 *The following safety*[1] *property must hold on the railroad crossing example: the gate is never open when the train is on the railroad crossing, or equivalently the gate is never open between the moment when the controller commands its lowering and the moment when it receives a signal exit from the train (P_1). In MITL syntax, P_1 is written $\Box(c_2 \Rightarrow \neg is\_up)$. The liveness*[2] *property: the gate is closed within the two t.u. after the controller received an approach signal from the train (P_2) is expressed by $\Box(is\_up \wedge c_1 \Rightarrow \Diamond_{<2} is\_down)$.*

3 Properties preservation using timed τ-simulations

Consider the railroad crossing example. Both properties P_1 and P_2 concern the behaviour of two components of the system, namely the gate and the controller. Therefore, it seems interesting to verify them only on these two components, instead of performing the verification on the complete model, obtained by the integration of one or several components *train* to these two components. Indeed, the assembling of the gate and the controller leads to a smaller-sized system than the whole one, and thus model-checking is more applicable.

However, by performing such a verification, it must be ensured that properties established on the composition *gate/controller* are preserved when integrating the component(s) *train*. A way to ensure such a preservation is to compare the behaviour of both models, i.e., the one on which verification is run, and the one on which properties must be preserved. A way to make such a comparison is to use equivalence or preorder relations. Equivalence relations are generally not adapted to incremental development, contrary to preorders. A preorder already used in the untimed case to guarantee preservation of properties during

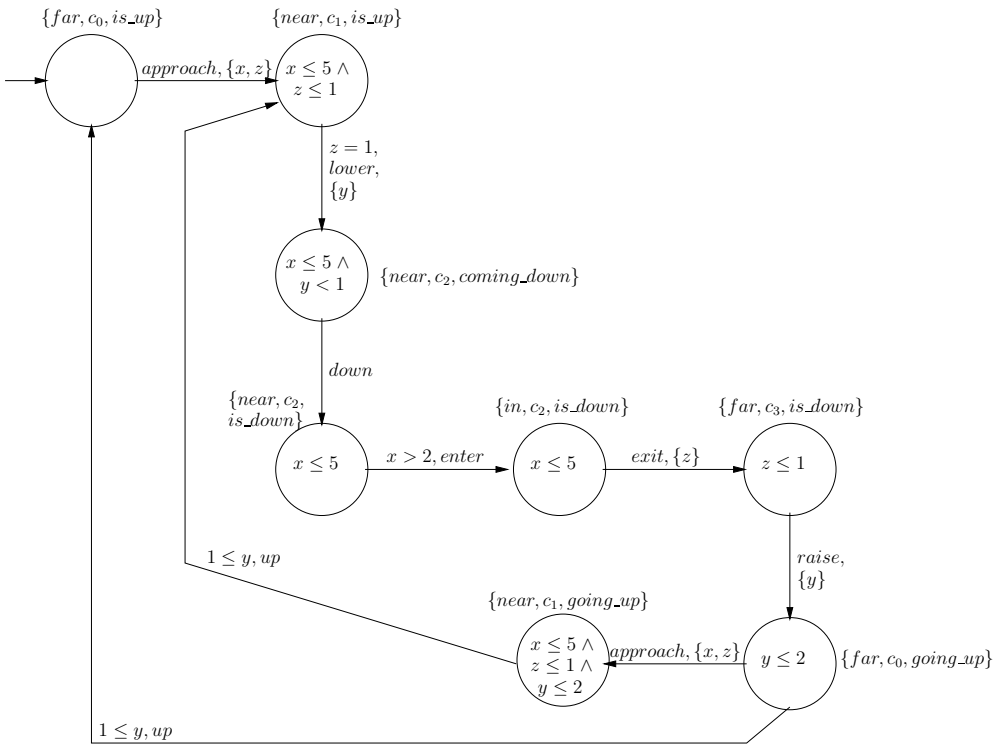

Figure 3 Parallel composition of the train, the gate and the controller

a refinement process for transitions systems is τ-simulation (Bellegarde et al. 2000). We extend this relation to take into account timing informations. By doing so, we obtain two τ-simulation relations: a timed τ-simulation dealing with the preservation of safety properties, and a stability-respecting and divergence-sensitive timed τ-simulation to handle the preservation of all MITL properties, in particular liveness ones[3].

3.1 Timed τ-simulation / safety properties

Consider two TA $A_1 = \langle Q_1, q_{0_1}, \Sigma_1, X_1, T_1, \texttt{Invar}_1, L_1\rangle$ and $A_2 = \langle Q_2, q_{0_2}, \Sigma_1 \cup \{\tau\}, X_2, T_2, \texttt{Invar}_2, L_2\rangle$ such that A_2 is obtained from A_1 either by refinement or by integration of components. Labels in $\Sigma_2 \backslash \Sigma_1$ concern actions introduced by the development process and are considered as being non-observable and renamed by τ. Thus, $\Sigma_2 = \Sigma_1 \cup \{\tau\}$. Other actions are called observable. The timed τ-simulation, called $\mathcal{S}$, is defined informally by the following points:

(i) If A_2 can make an observable action after some amount of time, then A_1 could do the same observable action after the same amount of time. In particular, this implies that observable actions can not be taken later in A_2 than they could be in A_1 (items 1, 2 and 5 of Definition 4).

(ii) Non-observable actions stutter (item 3 of Definition 4).

These points are illustrated in Fig. 4, where s_1 and s_1' are states of the semantic graph A_1, s_2 and s_2' are states of the semantic graph A_2, a is a label of A_1 and t is a time delay.

We also use a so-called gluing predicate, defined at a syntactic level on the atomic propositions of A_2 and A_1. This predicate is defined by the following grammar, where ap_1 and ap_2 are respectively atomic propositions of A_1 and A_2:

$$P_g \ ::= \ ap_1 \mid ap_2 \mid \neg p \mid p \vee p.$$

This gluing predicate induces a relation between locations of A_2 and A_1. That is, two locations $q_2 \in Q_2$ and $q_1 \in Q_1$ are in relation w.r.t. the gluing predicate P_g if they respect P_g, written $(q_1, q_2) \models_g P_g$, where:

$$(q_2, q_1) \models_g P_g \ \ \textit{iff} \bigwedge_{ap_2 \in L_2(q_2)} ap_2 \wedge P_g \Rightarrow \bigwedge_{ap_1 \in L_1(q_1)} ap_1.$$

Definition 4 (Timed τ-simulation) *Let* $A_1 = \langle \mathrm{Q}_1, \mathrm{q}_{0_1}, \Sigma_1, \mathrm{X}_1, \mathrm{T}_1, \texttt{Invar}_1, \mathrm{L}_1\rangle$ *and* $A_2 = \langle \mathrm{Q}_2, \mathrm{q}_{0_2}, \Sigma_1 \cup \{\tau\}, \mathrm{X}_2, \mathrm{T}_2, \texttt{Invar}_2, \mathrm{L}_2\rangle$ *be two timed automata s.t.* $X_1 \subseteq X_2$. *We call* S_1 *and* S_2 *the respective set of states of the semantic graphs* A_1 *and* A_2, *and* P_g *the gluing predicate provided between* A_2 *and* A_1. *The timed* τ*-simulation* $\mathcal{S}$ *is the greatest binary relation included in* $S_2 \times S_1$. *We say that* $(q_2, v_2)\mathcal{S}(q_1, v_1)$ *if the following conditions hold:*

1. *Strict simulation:*
$(q_2, v_2) \xrightarrow{e_2} (q_2', v_2') \wedge \texttt{label}(e_2) \in \Sigma_1 \ \Rightarrow$

$$\exists (q_1', v_1') \cdot ((q_1, v_1) \xrightarrow{e_1} (q_1', v_1') \ \wedge \ \texttt{label}(e_1) = \texttt{label}(e_2) \ \wedge \ (q_2', v_2') \ \mathcal{S} \ (q_1', v_1')).$$

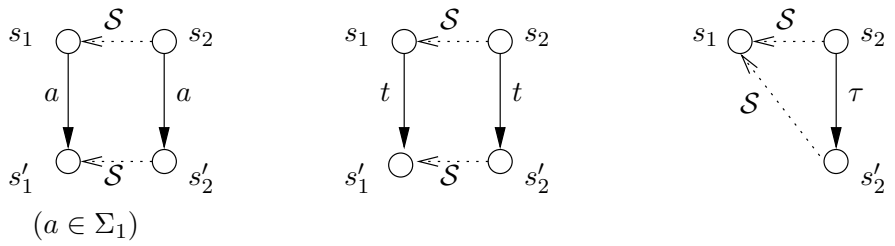

Figure 4 Illustration of timed τ-simulation

2. *Delay equality*[4]*:*
$(q_2, v_2) \xrightarrow{t} (q_2, v_2 + t) \Rightarrow \exists (q_1, v_1 + t) \cdot ((q_1, v_1) \xrightarrow{t} (q_1, v_1 + t) \wedge (q_2, v_2 + t)\ \mathcal{S}\ (q_1, v_1 + t))$.

3. *τ-transitions stuttering:*
$(q_2, v_2) \xrightarrow{e_2} (q'_2, v'_2) \wedge \mathtt{label}(e_2) = \tau \Rightarrow (q'_2, v'_2)\ \mathcal{S}\ (q_1, v_1)$.

4. *Location labelling respect:*
$(q_2, q_1) \models_g P_g$.

5. *Common clock valuation equality:*
$v_2\rfloor_{X_1} = v_1$.

We extend this notion of simulation to timed automata. Given two TA A_1 and A_2, and their respective initial state s_{0_1} and s_{0_2}, we say that A_1 simulates A_2 w.r.t. $\mathcal{S}$, written $A_2 \preceq_{\mathcal{S}} A_1$ if $s_{0_2} \mathcal{S} s_{0_1}$.

Remark 2 *In Definition 4, we consider the greatest relation included in $S_2 \times S_1$ and satisfying the clauses 1 to 5. Note that such a relation exists. Indeed, let us consider two relations R_1 and R_2, both included in $S_2 \times S_1$, satisfying the previous conditions. Trivially, the union of these two relations satisfy these conditions as well. Consider $\mathcal{S}$ as the union of all such relations included in $S_2 \times S_1$ and satisfying the conditions. Thus, $\mathcal{S}$ contains them all. Therefore, it is the greatest relation included in $S_2 \times S_1$ satisfying the conditions.*

Remark 3 *The following conditions are necessary syntactic conditions on A_2 and A_1 for the verification of the timed τ-simulation to succeed. First, consider an observable action a modeled by an edge $e_2 = (q_2, g_2, a, r_2, q'_2)$ in A_2, and by the edge $e_1 = (q_1, g_1, a, r_1, q'_1)$ in A_1. The edge e_2 must reset the same clocks in X_1 than e_1, i.e., $r_2 \cap X_1 = r_1$. Secondly, non observable actions in A_2 must only reset clocks which do not exist in A_1. Moreover, their guard must only involve such clocks. Formally, given a non-observable action in A_2 modeled by an edge $e = (q, g, \tau, r, q')$, we impose that $r \cap X_1 = \emptyset$ and that $g \in \mathcal{C}_{df}(X_2)$.*

It is well-known that such a simulation relation only preserves safety properties. To deal with the preservation of all MITL, in particular liveness, we restrict the relation with two additional clauses, called *divergence-sensitivity* and *stability-respect*.

3.2 Divergence-sensitive and stability-respecting timed τ-simulation / MITL properties

The relation $\mathcal{S}$ between two TA A_2 and A_1 guarantees that the sequences of observable actions of A_2, with possibly τ-actions inserted, are sequences of actions which also exist in A_1, and that each observable action in A_2 occurs at most after the same time delay than in A_1.
Consider the liveness property P_2, expressed by $\Box(is\_up \wedge c_1 \Rightarrow \Diamond_{<2} is\_down)$, and recall that it concerns the composition of the components *gate* and *controller*. Intuitively, for this property to be preserved when adding the component *train*, runs of the obtained model must not be cut between the moment when $is\_up \wedge c_1$ holds and the moment when *is_down* is reached. However, during composition, sequences of observable actions (i.e., actions of the gate and the controller) can be cut, either by introducing a deadlock when adding the component *train* or by the introduction of an infinite sequence of non-observable actions. Thus, to preserve such a property, the integration of the component *train* must not introduce deadlocks neither infinite sequences of non-observable actions. These two criteria are respectively called *stability-respect* and *divergence-sensitivity* (Glabbeck 1993).

Stability-respect. To express this criterion, we use the predicate `free` defined in (Tripakis 1998) (`free` stands for *deadlock-free*). Informally, given a location q, $\mathtt{free}(q)$ is the set of all valuations (of states with q as discrete part) from which a discrete transition can be taken after some time elapsed. In other words, it is the set of all valuations for which the location is not a deadlock. The formal definition is:

$$\mathtt{free}(q) = \bigcup_{e=(q,g,a,r,q') \in T} \swarrow (g \cap ([r := 0]\mathtt{Invar}(q'))).$$

Divergence-sensitivity. The detection of infinite sequences of non-observable actions in a timed automaton A consists in detecting non-zeno τ-cycles in A (a cycle which only contains timed transitions and discrete transitions labelled by τ is called a τ-cycle). Formally, we say that A does not contain any non-zeno τ-cycles if:

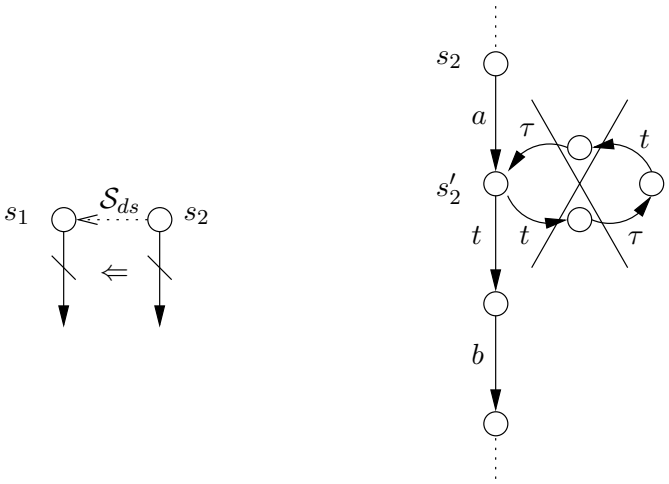

Figure 5 Stability-respect and divergence-sensitivity

$$\forall \rho, k \cdot (\rho \in \Gamma(A) \wedge \texttt{time}(\rho) = \infty \wedge k \geq 0 \Rightarrow \exists k', e \cdot (k' \geq k \wedge (\rho, k') \xrightarrow{e} (\rho, k'+1) \wedge \texttt{label}(e) \neq \tau)).$$

These two notions are illustrated in Fig. 5, where s_1 is a state of the semantic graph A_1, s_2 and s_2' are states of the semantic graph A_2 and a and b are common labels between A_1 and A_2.

Definition 5 ***(Divergence-sensitive (DS) timed τ-simulation)*** *Consider two TA $A_1 = \langle \mathrm{Q}_1, \mathrm{q}_{0_1}, \Sigma_1, \mathrm{X}_1, \mathrm{T}_1, \texttt{Invar}_1, \mathrm{L}_1 \rangle$ and $A_2 = \langle \mathrm{Q}_2, \mathrm{q}_{0_2}, \Sigma_1 \cup \{\tau\}, \mathrm{X}_2, \mathrm{T}_2, \texttt{Invar}_2, \mathrm{L}_2 \rangle$, such that $X_1 \subseteq X_2$. S_1 and S_2 are the respective set of states of A_1 and A_2. The relation $\mathcal{S}_{ds}$ is included in $S_2 \times S_1$. We say that $(q_2, v_2)\mathcal{S}_{ds}(q_1, v_1)$ if $(q_2, v_2)\mathcal{S}(q_1, v_1)$ and*

- *Stability-respect: $v_2 \notin \texttt{free}(q_2) \Rightarrow v_1 \notin \texttt{free}(q_1)$,*
- *Divergence-sensitivity: A_2 does not contain any non-zeno τ-cycles.*

Let A_1 and A_2 be two TA, with respective initial state s_{0_1} and s_{0_2}. We say that A_2 simulates A_1 w.r.t. $\mathcal{S}_{ds}$, written $A_2 \preceq_{\mathcal{S}_{ds}} A_1$, if $s_{0_2}\ \mathcal{S}_{ds}\ s_{0_1}$.

3.3 Preservation of properties

We defined the relation $\mathcal{S}_{ds}$ to preserve a larger spectrum of properties. We prove in this section that the relation preserves all MITL properties, as well as strong non-zenoness and deadlock-freedom.

3.3.1 MITL.

Consider two TA A_1 and A_2 such that $A_2 \preceq_{\mathcal{S}_{ds}} A_1$. We prove that, for each run in A_2, if the run of A_1 which simulates it satisfies a MITL property φ, then this run of A_2 also satisfies φ. Before proving this, recall that the satisfaction of a MITL formula is not defined on runs, but on timed state sequences. Thus, the following result is necessary to prove the preservation.

Lemma 1 *Consider two runs ρ_1 and ρ_2, such that $\rho_2 \preceq_{\mathcal{S}_{ds}} \rho_1$. Let σ_1 and σ_2 be the timed state sequences in which are respectively inscribed ρ_1 and ρ_2. Consider $t \in \mathbb{R}^+$, and the suffixes σ_1^t and σ_2^t of σ_1 and σ_2 at time t. We have:*

$$\forall \rho_2' \text{ inscribed in } \sigma_2^t, \exists \rho_1' \text{ inscribed in } \sigma_1^t \text{ such that } \rho_2' \preceq_{\mathcal{S}_{ds}} \rho_1'.$$

Proof. Let us consider ρ_2' as the suffix of ρ_2 at time t. We distinguish two cases:

1. There exists a state (q_2, v_2) in ρ_2 s.t. $\texttt{time}(\rho_2, (q_2, v_2)) = t$. As $\rho_2 \preceq_{\mathcal{S}_{ds}} \rho_1$, there exists (q_1, v_1) in ρ_1 s.t. $(q_2, v_2)\mathcal{S}_{ds}(q_1, v_1)$. By clause 2 of Def. 4, we can deduce that $\texttt{time}(\rho_1, (q_1, v_1)) = t$. Let ρ_1' be the suffix of ρ_1 from the state (q_1, v_1). We have $\rho_2' \preceq_{\mathcal{S}_{ds}} \rho_1'$.
2. If such a state (q_2, v_2) does not exist in ρ_2, it means that this time value occurs during a time transition. Then, it is enough to split the time transition and create an intermediary state (q_2, v_2) s.t. $\texttt{time}(\rho_2, (q_2, v_2)) = t$. Then, if we also split the corresponding time transition in ρ_1 (this transition exists since $\rho_2 \preceq_{\mathcal{S}_{ds}} \rho_1$), the previous case applies.

The lemma is true for the particular case of the suffix ρ_2^t of run ρ_2 at time t built from σ_2^t. Without loss of generality, we can affirm that the lemma is also true for each run inscribed in σ_2^t (as each run can be written as ρ_2^t by splitting or concatenating time transitions). □

Recall that the locations of A_2 are not labelled over the same set of propositions (called $Props_2$) than the locations of A_1 (called $Props_1$). A gluing predicate is defined between A_2 and A_1 to establish a correspondence between propositions of A_2 and propositions of A_1. As MITL properties of A_1 are expressed over $Props_1$, they will be satisfied (by preservation) on A_2 modulo the gluing predicate P_g. The satisfaction by preservation of a MITL formula φ over $Props_1$, written $\sigma \models_p \varphi$, is defined as follows (where σ is a TSS of A_2, and ap, φ, ψ and ϕ are MITL formulas over $Props_1$):

- $\sigma \models_p \texttt{true}$ is true,
- $\sigma \models_p ap$ iff $\bigwedge_{ap_2 \in L_2(\texttt{disc}((\sigma,0)))} ap_2 \wedge P_g \Rightarrow ap$,
- $\sigma \models_p \neg\varphi$ iff it is not true that $\sigma \models_p \varphi$,
- $\sigma \models_p \phi \vee \psi$ iff $\sigma \models_p \phi$ or $\sigma \models_p \psi$,
- $\sigma \models_p \phi\ \mathcal{U}_I\ \psi$ iff there exists $t \in I$ such that $\sigma^t \models_p \psi$, and $\forall t' \in]0, t[$, $\sigma^{t'} \models_p \phi$.

By extension, a TA A satisfies a MITL property φ by preservation if all its runs satisfies φ by preservation:

$$\forall \rho \cdot (\rho \in \Gamma(A) \Rightarrow \sigma(\rho) \models_p \varphi).$$

Lemma 2 (Preservation of MITL properties on a run) *Let A_1 and A_2 be two TA such that $A_2 \preceq_{\mathcal{S}_{ds}} A_1$, ρ_1 a run of A_1 and ρ_2 a run of A_2. Consider a* MITL *property φ. We have:*

$$\rho_2 \preceq_{\mathcal{S}_{ds}} \rho_1 \wedge \sigma(\rho_1) \models \varphi \Rightarrow \sigma(\rho_2) \models_p \varphi.$$

Proof. We prove, by induction on the structure of the formula, that $\mathcal{S}_{ds}$ preserves MITL properties. In the following, for more readability, we write σ_1 for $\sigma(\rho_1)$ and σ_2 for $\sigma(\rho_2)$.
The basis, i.e., when the formula is an atomic proposition, comes directly from the fact that the relation respects the gluing predicate. The cases $\vee$ and $\neg$ are trivial. The interesting case is when the type of formula is $\varphi \mathcal{U}_I \psi$. Let us prove that this kind of formula is preserved.
Since $\sigma_1 \models \phi \,\mathcal{U}_I\, \psi$, there exists a suffix σ_1^t at time $t \in I$ such that $\sigma_1^t \models \psi$. Consider now the suffix σ_2^t of σ_2 at time t. The relation $\mathcal{S}_{ds}$ forbids infinite sequences only composed of non-observable actions in ρ_2 (and thus, in σ_2). Thus, each discrete action which occurs in ρ_1 (and in σ_1) also occurs in ρ_2 (and thus in σ_2). Moreover, since $\mathcal{S}_{ds}$ does not allow introduction of deadlocks in ρ_2, if time t can be reached in ρ_1 (and in σ_1), then this time value t can also be reached in ρ_2 (and in σ_2). Consequently, if the suffix σ_1^t exists, the suffix σ_2^t also exists.
We now prove that σ_2^t satisfies ψ. We know that σ_1^t satisfies ψ and that (induction hypothesis) ψ is preserved. It remains to prove that each run inscribed in σ_2^t is in relation w.r.t. $\preceq_{\mathcal{S}_{ds}}$ with a run inscribed in σ_1^t. By lemma 1, each run ρ_2' inscribed in σ_2^t is in relation (w.r.t $\preceq_{\mathcal{S}_{ds}}$) with a run ρ_1' inscribed in σ_1^t. Thus we have that $\sigma_2^t \models_p \psi$.
Since $\sigma_1 \models \phi \,\mathcal{U}_I\, \psi$, then $\forall t' \in]0, t[$, $\sigma_1^{t'} \models \phi$. And thus, as previously, with lemma 1 and the induction hypothesis that ϕ is preserved, $\forall t' \in]0, t[$, $\sigma_2^{t'} \models_p \phi$.
Thus, $\sigma_2 \models_p \phi \,\mathcal{U}_I\, \psi$.
□

Theorem 1 (Preservation of MITL properties) *Let φ be a* MITL *formula, A_1 and A_2 be two TA. If $A_1 \models \varphi$ and $A_2 \preceq_{\mathcal{S}_{ds}} A_1$ then $A_2 \models_p \varphi$.*

Proof. The proof is immediate. Since $A_2 \preceq_{\mathcal{S}_{ds}} A_1$, then $\forall \rho_2 \cdot (\rho_2 \in \Gamma(A_2) \Rightarrow \exists \rho_1 \cdot (\rho_1 \in \Gamma(A_1) \wedge \rho_2 \preceq_{\mathcal{S}_{ds}} \rho_1))$.
Since $A_1 \models \varphi$, then all its runs also satisfy this property, i.e., $\forall \rho_1 \cdot (\rho_1 \in \Gamma(A_1) \Rightarrow \sigma(\rho_1) \models \varphi)$. By lemma 2, φ also holds by preservation, for all runs of A_2, and thus $A_2 \models_p \varphi$.
□

3.3.2 Strong non-zenoness.

We prove that $\mathcal{S}_{ds}$ preserves strong non-zenoness.

Proposition 1 *Consider two TA A_1 and A_2. If A_1 is strongly non-zeno and $A_2 \preceq_{\mathcal{S}_{ds}} A_1$, then A_2 is strongly non-zeno.*

Proof. *(sketch, by contradiction).* We prove intuitively this proposition. Consider that A_1 is strongly non-zeno, and that A_2 is not. Thus, A_2 contains a run ρ_2 which is zeno. As $A_2 \preceq_{\mathcal{S}_{ds}} A_1$, then each infinite run of A_2 is simulated by an infinite run of A_1. Let ρ_1 be the run which simulates ρ_2. The clause *delays equality* ensures that if the total time elapsed in ρ_2 converges, then the total time elapsed in ρ_1 also converges. This is contradictory with the assumption that A_1 is strongly non-zeno, and thus that it does not contain any executions in which the total time elapsed converges.
Recall that the run ρ_2 contains the same sequence of observable actions than ρ_1, with also non-observable actions which can be inserted between observable actions. As the time delay between two observable actions must remain the same than in A_1, if an infinite number of non-observable actions is inserted in this finite delay, then the run becomes zeno. But, this case is excluded by the clause *divergence-sensitivity*, which forbids such infinite sequences of non-observable actions.
□

Remark 4 (Non-zenoness preservation) *The weaker notion of non-zenoness is not preserved by the DS timed τ-simulation. Recall that non-zenoness is the fact that there is not reachable state which is zeno. In other words, non-zenoness expresses that there is no reachable state such that all runs leaving from it are zeno. Therefore, contrary to strong non-zenoness which imposes that all runs are non-zeno, non-zenoness only requires that there exists at least one run from each reachable state.*
In terms of runs, the simulations we defined express that each run in A_2 is simulated by a run in A_1. It is thus possible that some runs in A_1 do not simulate any runs in A_2. These runs somehow do not exist in A_2. Consider now a state s_1 in A_1 from which only one run is non-zeno, and a state s_2 in relation with s_1. It can be the case that the non-zeno run from s_1 does not exist any more from s_2, and thus that only zeno runs leave from s_2. Non-zenoness is thus immediatly not preserved.

3.3.3 Deadlock-freedom.

Deadlock-freedom is preserved by the DS timed τ-simulation, as a direct consequence of the definition of the relation (with the clause *stability-respect*).

Proposition 2 *Let A_1 and A_2 be TA. If A_1 is deadlock-free and $A_2 \preceq_{\mathcal{S}_{ds}} A_1$, then A_2 is deadlock-free.*

4 Exploiting simulations for incremental development

In the previous section, we defined two relations: a timed τ-simulation which preserves safety properties, and a

divergence-sensitive and stability-respecting timed τ-simulation, which preserves all MITL properties, strong non-zenoness and deadlock-freedom. To exploit these relations, and thus their preservation abilities, during an incremental development of a component-based timed system, it is necessary to study the three following properties:

- compatibility of the relations w.r.t. composition operators,
- composability, which means that a component simulates its integration with other components,
- and compositionality. This last property expresses that, given components A, B, C and D, if B simulates A and D simulates C, then the composition of B and D simulates the composition of A and C.

Composability is essential for integration of components since it guarantees automatically the preservation of properties during the integration (the kind of properties preserved depends on the notion of simulation considered). Compositionality is essential for refinement in order to verify it in a compositional way.

We study these properties in the case of the classic parallel composition. In the sequel, we use the following notations. Given a timed automaton A, we note S_A its set of states and Σ_A its alphabet. A state of A is simply written s_A or s'_A, which respectively represents the pairs (q_A, v_A) or (q'_A, v'_A). The initial state of A is written s_{0_A}.

Proposition 3 (Composability) *Let A and B be TA. We have: $A\|B \preceq_{\mathcal{S}} A$.*

Proof. By construction of $A\|B$, its initial state is the pair (s_{0_A}, s_{0_B}). To prove that $A\|B \preceq_{\mathcal{S}} A$, it is enough to prove that $(s_{0_A}, s_{0_B})\mathcal{S}s_{0_A}$. By definition, $\preceq_{\mathcal{S}}$ is the greatest relation included in $S_{A\|B} \times S_A$ which satisfies clauses 1 to 3 of Definition 4. Thus, each relation $R \subseteq S_{A\|B} \times S_A$ which satisfies these clauses is included in $\preceq_{\mathcal{S}}$. Consider a relation $R \subseteq S_{A\|B} \times S_A$ such that $\forall (s_A, s_B) \in S_{A\|B}$,

$$(s_A, s_B) R\ s'_A \text{ if } s_A = s'_A.$$

Consider $((s_A, s_B), s_A) \in R$.

1. *Strict simulation*: let $(s_A, s_B) \xrightarrow{a} (s'_A, s'_B)$ in $A\|B$ such that $a \in \Sigma_A$. By construction of $A\|B$, a transition $s_A \xrightarrow{a} s'_A$ exists in A. By definition of R, $(s'_A, s'_B) R\ s'_A$ and R satisfies the *strict simulation*.
2. *Delay equality*: same arguments than those for strict simulation can be used to prove that this clause holds for R.
3. *τ-transitions stuttering*: consider a transition $(s_A, s_B) \xrightarrow{\tau} (s'_A, s'_B)$ in $A\|B$. Recall that τ-transitions represent non-observable actions initially labelled by an action in $\Sigma_B \backslash \Sigma_A$. By construction of $A\|B$, $s'_A = s_A$. Thus, $(s_A, s'_B) R\ s_A$ and R satisfies *τ-transitions stuttering*.
4. *Location labelling respect*: immediate by definition of R.
5. *Common clocks valuation equality*: immediate by definition of R and since X_A and X_B are disjoint.

□

Proposition 4 (Compatibility) *Let A, B and C be TA. If $A \preceq_{\mathcal{S}} B$ then $A\|C \preceq_{\mathcal{S}} B\|C$.*

Proof. The structure of the proof is similar to the previous one. Details can be found in Appendix A.
□

Proposition 5 (Compositionality) *Let A, B, C, D be TA. If $A \preceq_{\mathcal{S}} B$ and $C \preceq_{\mathcal{S}} D$ then $A\|C \preceq_{\mathcal{S}} B\|D$.*

Proof. Immediate with Proposition 4. Since $A \preceq_{\mathcal{S}} B$, then $A\|C \preceq_{\mathcal{S}} B\|C$. Since $C \preceq_{\mathcal{S}} D$, then $B\|C \preceq_{\mathcal{S}} B\|D$. By transitivity of the relation $\preceq_{\mathcal{S}}$, we have $A\|C \preceq_{\mathcal{S}} B\|D$.
□

The timed τ-simulation is well-adapted to incremental development with the classic parallel composition operator, since composability, compatibility and compositionality hold for free. This is not the case for the DS timed τ-simulation when using such a composition paradigm. Indeed, the fact that this kind of composition generally introduces deadlocks does not allow to benefit of the three properties. For composability for instance, this introduction of deadlocks is incompatible with the stability-respect clause of the DS timed τ-simulation. However, it remains possible to benefit of its assets in terms of preservation, but at the cost of an algorithmic verification of the relation.

5 Simulations in practice

In the previous sections, we defined the simulations at a semantic level, on the states of TA. To check algorithmically the DS timed τ-simulation, we extend it on the symbolic representation of TA, i.e., on zones.

5.1 Symbolic timed τ-simulations

We focus directly on the symbolic version of the DS timed τ-simulation. The definition for the timed τ-simulation can be obtained by not considering divergence-sensitivity and stability-respect.
Most of the clauses of this symbolic definition can be obtained straightforward from the semantic definition. The main changes concern the clauses *delays equality*, *common clocks valuations equality* and *strict simulation*. Consider two simulation graphs SG_1 and SG_2, z_1 a zone of SG_1 and z_2 a zone of SG_2. In simulation graphs, time elapsing does not appear explicitly as

transitions. Intuitively, time elapses inside zones. Thus, delays equality and common clocks valuations equality are checked by verifying that the polyhedron of the zone z_2 (projected on the set of clocks of SG_1) is included in the polyhedron of z_1.
Let us now study strict simulation. The definition we give at the symbolic level is based on the post-stability property of the simulation graph. This property ensures that, given a transition $z \xrightarrow{e} z'$, all the successors of (semantic) states in z and taking e are in z'. A part of this clause is directly extended from the semantic one, by imposing that if the transition $z_2 \xrightarrow{e_2} z_2'$ exists, then a transition $z_1 \xrightarrow{e_1} z_1'$ exists with the same label. But, as the simulation graph does not have the pre-stability property (i.e., a symbolic transition $z \xrightarrow{e} z'$ does not imply that all the (semantic) states in z can take the transition e), another condition is needed. Each (semantic) state in z_2 taking this transition e_2 must correspond to a state in z_1 taking transition e_1. This clause is illustrated by Fig. 6. To express this condition, we define a predicate named $\texttt{src\_val}$, defined as follows: $\texttt{src\_val}(z, e, z') = \texttt{poly}(\texttt{pre}(e, z') \cap z)$.

Definition 6 (Symbolic DS timed τ-simulation) *Let $SG_1 = \langle \mathrm{Z}_1, \mathrm{z}_{0_1}, \Sigma_1, \mathcal{T}_1 \rangle$ and $SG_2 = \langle \mathrm{Z}_2, \mathrm{z}_{0_2}, \Sigma_2, \mathcal{T}_2 \rangle$ be two simulation graphs, obtained respectively from two TA A_1 and A_2. Let X_1 be the set of clocks of A_1. The symbolic DS timed τ-simulation $\mathcal{Z}_{ds}$ is the greatest binary relation included in $Z_2 \times Z_1$, such that $z_2 \mathcal{Z}_{ds} z_1$ if the following conditions hold:*

1. *Strict simulation:*
$$z_2 \xrightarrow{e_2} z_2' \wedge \texttt{label}(e_2) \in \Sigma_1 \Rightarrow \exists z_1' \cdot (z_1 \xrightarrow{e_1} z_1' \wedge \texttt{label}(e_1) = \texttt{label}(e_2) \wedge \texttt{src\_val}(z_2, e_2, z_2')\rfloor_{X_1} \subseteq \texttt{src\_val}(z_1, e_1, z_1') \wedge z_2' \ \mathcal{Z}_{ds} \ z_1').$$

2. *Delay equality and common clock valuation equality:*
$\texttt{poly}(z_2)\rfloor_{X_1} \subseteq \texttt{poly}(z_1)$.

3. *τ-transition stuttering:*
$z_2 \xrightarrow{e_2} z_2' \wedge \texttt{label}(e_2) = \tau \Rightarrow z_2' \ \mathcal{Z}_{ds} \ z_1$.

4. *Location labelling respect:*
$(\texttt{disc}(z_2), \texttt{disc}(z_1)) \models_g P_g$.

5. *Stability-respect:*
$(\texttt{poly}(z_2) \backslash \texttt{free}(\texttt{disc}(z_2))\rfloor_{X_1} \subseteq \texttt{poly}(z_1) \backslash \texttt{free}(\texttt{disc}(z_1))$.

6. *Divergence-sensitivity:*
SG_2 does not contain any non-zeno τ-cycles[5].

We extend this relation to simulation graphs. Consider two simulation graphs SG_1 and SG_2. Their initial zones are respectively z_{0_1} and z_{0_2}. We say that SG_1 simulates SG_2 w.r.t. $\mathcal{Z}_{ds}$, written $SG_2 \preceq_{\mathcal{Z}_{ds}} SG_1$, if $z_{0_2} \mathcal{Z}_{ds} z_{0_1}$.

We now prove that this symbolic relation $\mathcal{Z}_{ds}$ implies the semantic relation $\mathcal{S}_{ds}$. This implication allows to benefit of the preservation abilities of $\mathcal{S}_{ds}$ at the symbolic level. The proof is decomposed into two parts. First, we express Lemma 3, which is necessary for the divergence-sensitivity clause, then we prove the implication of the relations in Lemma 4.

Lemma 3 *Let A be a TA and c be the greatest constant appearing in a constraint of A. Let $SG(A, c)$ be the simulation graph associated to A. If A does not contain any non-zeno τ-cycles, then $SG(A, c)$ does not contain any non-zeno τ-cycles, and conversely.*

Proof. Immediate by the fact that each non-zeno run of A is inscribed in a unique non-zeno path of $SG(A, c)$, and that for each non-zeno path of $SG(A, c)$, there exists a non-zeno run inscribed in this path.
□

Lemma 4 *Let SG_1 and SG_2 be two simulation graphs. Let (q_1, ζ_1) and (q_2, ζ_2) be two respective zones of SG_1 and SG_2, such that $(q_2, \zeta_2)\mathcal{Z}_{ds}(q_1, \zeta_1)$. We have the following. For each state $(q_2, v_2) \in (q_2, \zeta_2)$, there exists only one state $(q_1, v_1) \in (q_1, \zeta_1)$ such that $(q_2, v_2)\mathcal{S}_{ds}(q_1, v_1)$. Formally:*

$$(q_2, \zeta_2) \ \mathcal{Z}_{ds} \ (q_1, \zeta_1) \Rightarrow \forall v_2 \cdot (v_2 \in \zeta_2 \Rightarrow \exists v_1 \cdot (v_1 \in \zeta_1 \wedge (q_2, v_2) \ \mathcal{S}_{ds} \ (q_1, v_1))). \quad (\star)$$

Proof. The proof is done by fixed-point induction. Consider the function $\mathcal{F} : Z_2 \times Z_1 \to Z_2 \times Z_1$ defined in the following way. If $\mathcal{Z}_{ds} \subseteq Z_2 \times Z_1$, then $((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds})$ iff the clauses 1 to 6 of Definition 6 hold.

In the definition of $\preceq_{\mathcal{Z}_{ds}}$, we consider the greatest fixed-point of the function $\mathcal{F}$. This function is trivially monotonic (and the sets Z_2 and Z_1 are finite), thus we can reason inductively about the pre-fixed-points of this function. The principle of this induction is the following: given a predicate P, this induction allows to prove $P(\mathcal{Z}_{ds})$, by proving $P(\mathcal{F}(\mathcal{Z}_{ds}))$, with the induction assumption that $P(\mathcal{Z}_{ds})$ holds. We proceed clause by clause. Details are given in Appendix B.
□

Theorem 2 *Let A_1 and A_2 be two TA, and SG_1 and SG_2 the simulation graphs respectively obtained from A_1 and A_2. If $SG_2 \preceq_{\mathcal{Z}_{ds}} SG_1$ then $A_2 \preceq_{\mathcal{S}_{ds}} A_1$.*

Proof. Since $SG_2 \preceq_{\mathcal{Z}_{ds}} SG_1$, we have $z_{0_2} \ \mathcal{Z}_{ds} \ z_{0_1}$. Let s_{0_2} and s_{0_1} be the respective initial states of A_2 and A_1. By definition, $s_{0_2} \in z_{0_2}$ and $s_{0_1} \in z_{0_1}$. By lemma 4, s_{0_2} is in relation (w.r.t. $\mathcal{S}_{ds}$) with a unique state s in z_{0_1}. By definition of $\mathcal{S}_{ds}$, the valuations of s and s_{0_2} over X_1 are equal. It follows that $s = s_{0_1}$ and $s_{0_2}\mathcal{S}_{ds}s_{0_1}$.
□

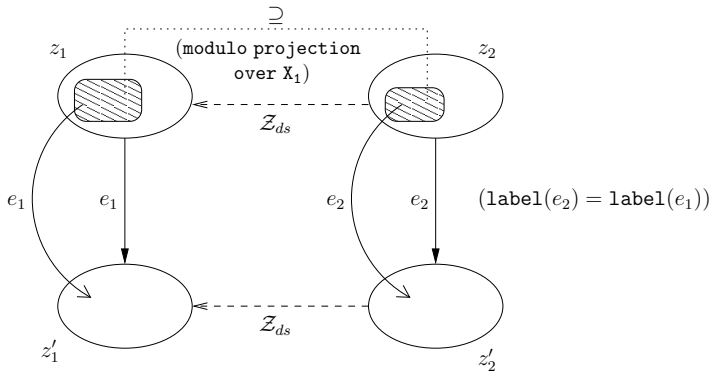

Figure 6 Clause *strict simulation* at the symbolic level

5.2 The tool VeSTA

We implemented the verification of the DS timed τ-simulation $\mathcal{Z}_{ds}$ in a tool named VeSTA[6](**Ve**rification of **S**imulations for **T**imed **A**utomata) (Bellegarde, Julliand, Mountassir & Oudot 2006*b*). VeSTA considers component-based timed systems developed incrementally using timed automata and the classic parallel composition operator. The tool focuses on the verification of the simulation during incremental development achieved by integration of components using the classic composition $\|$. Thus, it allows to ensure that local properties of a component (or a group of components) are preserved when it is merged with other components.
Consider two TA A_1 and A_2. To check if $A_1 \| A_2 \preceq_{\mathcal{Z}_{ds}} A_1$, two main algorithms are implemented in the tool. The first algorithm performs a joint on-the-fly depth-first search of the simulation graphs of A_1 and A_2 and checks the stability-respecting timed τ-simulation (i.e., clauses 1 to 5 of $\mathcal{Z}_{ds}$). If the verification of this part of the simulation does not succeed, the tool reports a diagnostic. This diagnostic consists in a symbolic trace in A_2 with a zone which does not satisfy the relation, and the corresponding symbolic trace in A_1. The divergence-sensitivity part consists in the detection of non-zeno τ-cycles in A_2. Thus, we reuse the module PROFOUNDER(Tripakis, Yovine & Bouajjani 2005), which is part of the OPEN-KRONOStool(Tripakis 1998) to achieve this search. PROFOUNDER was initially designed to check if a state is reachable in a TA, or to check timed Büchi automata emptiness. This second possibility consists in detecting non-zeno cycles in a TA. We adapted it to search only non-zeno τ-cycles.

5.3 A case study: the production cell

We present in this section[7] a case study on which we performed experiments to show the suitability of timed τ-simulations for incremental development, even when an algorithmic verification of the simulation is needed. The verification of the properties was achieved with the tool KRONOS (Yovine 1997). KRONOS is a verification tool for timed systems which performs TCTL model-checking (Alur, Courcoubetis & Dill 1993), in particular for component-based timed models (indeed, KRONOS can compute the parallel composition of TA). TCTL is a logical formalism that allows to express branching-time properties. It can be seen as the timed extension of the untimed logic CTL (Clarke & Emerson 1981, Emerson & Halpern 1982). To our knowledge, there is no tool performing MITL model-checking. Thus, we focused on linear-time properties that can also be expressed in TCTL to perform the verification with KRONOS. The verification of the simulations was achieved with VeSTA.

The production cell case study was developed by FZI (the Research Center for Information Technologies, in Karlsruhe) as part of the Korso project. The goal was to study the impact of the use of formal methods when treating industrial applications. Thus, this case study was treated in about thirty different formalisms. We treat it with timed automata, as it was in (Burns 2003).

5.3.1 Modeling.

The cell is made up of six devices, where pieces are treated: a feed-belt, equipped with a sensor, where pieces arrive, a deposit-belt from which pieces are evacuated, an elevating-rotary table, a two-arms robot and a press. The sensor detects when a piece is introduced in the system, and sends a signal to the robot to inform it of this arrival. When the piece is at the end of the feed-belt, it is transferred to the table which goes up and turns until being in a position from which the arm A of the robot can take the piece. The robot turns 90° so that the arm A can put down the piece on the press, where it is processed and then transported by the arm B to the deposit belt.
The cell is presented in Fig. 7. It is modeled by at least seven components: one for each device, and one or several pieces. The cell is subject to timing constraints, which are shown in Fig. 8. Each component is modeled as a TA. These TA can be found in (Burns 2003). The complete model is obtained by making the classic parallel composition of all these components.

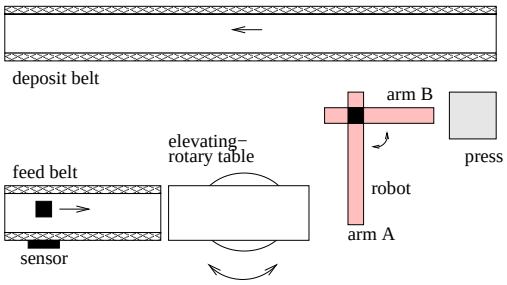

Figure 7 The Production Cell

Device	Description	Time
Robot	moves to press	5
Robot	turns 90°	15
Robot	moves to deposit belt	5
Robot	from deposit belt to table	25
Robot	from deposit belt to wait pos.	22
Robot	from press to wait pos.	17
Robot	from wait pos. to table	3
Robot	from wait pos. to press	2
Robot	at wait pos.	2
Feed Belt	piece moves to sensor	3
Feed Belt	piece moves to table	1
Table	raises and turns	2
Table	returns and turns	2
Press	presses a piece	22-25
Press	ready for a new piece	18-20
Deposit Belt	evacuates a piece	4

Figure 8 Timing constraints for the production cell

Component	Robot	Press	Feed belt	Dep. Belt
States/Trans.	39/40	7/7	6/6	4/4
Component	Table	Sensor	Piece	Complete Model
States/Trans.	6/6	2/2	7/7	1655/2395

Figure 9 Size of the simulation graphs of each component of the production cell

Fig. 9 shows the size of the simulation graphs for each component.

5.3.2 Verification.

To ensure that the modeling is correct, there are several properties to check. Recall that we focus on the local properties of the components (or group of components). We propose to perform the verification locally, and then to ensure that the properties hold on the global model by preservation, by checking algorithmically the DS timed τ-simulation. In particular, we consider local properties concerning the robot. Here is a non-exhaustive list of dynamic properties to check on this component. Properties 1 and 2 are safety requirements, properties 3 and 4 are liveness ones and properties 5 to 7 are bounded-response ones:

(1) When the robot is in wait position, its two arms are empty,

(2) The robot is not waiting in front of the table if the arm A is full,

(3) If there is a piece on arm B, the robot will eventually go to the deposit belt,

(4) If there is a piece on arm A, the robot will eventually go to the press,

(5) When the robot is in front of the deposit belt, then it goes back to the table within 25 t.u. if there are no pieces on the press,

(6) When the robot is in front of the deposit belt, then it goes to the wait position within 22 t.u. if there is a piece on the press,

(7) When it is in wait position, either the robot goes to the press within 2 t.u. to unload it or it goes back to the table within 3 t.u. to pick up a new piece.

The following liveness property concerns the correct interaction between the robot and the press :

(8) If arm A is full then the press will eventually be free.

We use two approaches to verify these properties on the plant. As a first approach, we verify the properties in a classic way, directly on the global model, with one piece. As a result, we obtain that all the properties hold on this model of the plant. The second approach consists in verifying the properties locally, i.e., on the robot component for properties 1 to 7, and on the composition $robot \| press$ for property 8. Here again, the verification succeeds. Next, to guarantee the preservation of these properties, first when integrating the robot with the press, then when integrating the composition $robot \| press$ with the rest of the components, we use the DS timed τ-simulation. More precisely, we check that

$$robot \| press \preceq_{\mathcal{Z}_{ds}} robot$$

to ensure that properties 1 to 7 are preserved on the composition $robot \| press$, and

$$complete\ model \preceq_{\mathcal{Z}_{ds}} robot \| press$$

to check the preservation of property 8 on the complete model (as well as the preservation of properties 1 to 7 if their preservation on $robot \| press$ is established). We use our tool VeSTA to check it for both cases, and obtain as a result that the verification of the simulation was successful. Thus, properties are preserved.
Fig. 10 gives the detailed results of the comparison of the two approaches in terms of verification times (in seconds). We can see that, even on this small example, the second approach only needs 0.57 sec. of computation time to ensure that the properties hold on the cell (0.06

Property	Global verification (Kronos)	Local verification (Kronos)	Preservation checking (VesTA)
Prop. 1	0.01	< 0.001	
Prop. 2	0.01	< 0.001	
Prop. 3	0.98	< 0.001	
Prop. 4	15.79	0.04	0.05
Prop. 5	0.68	< 0.001	
Prop. 6	0.48	< 0.001	
Prop. 7	0.7	< 0.001	
Prop. 8	0.93	0.02	0.46
Total	19.58	0.06	0.51

Figure 10 Production cell: local and global verification times (in seconds)

sec. for local verification and 0.51 sec. for preservation), whereas the classic approach consumes 19.58 sec.

Note that, in both approaches, we focused on a global system which contains only one piece. The reason is the following. First, in the case of the second approach, adding other pieces to the global system does not affect the results of the preservation. As the component piece already exists in the global system and that there are no synchronizations between the pieces, no new deadlocks can appear while adding a new piece. Indeed, the system can behave like it did with only one piece, or synchronize with the new piece. In this last case, as the environment of the pieces could synchronize with one piece without introducing deadlocks, then it will synchronize with the new pieces in the same way, thus without introducing deadlocks. On the other hand, in the first approach, adding pieces considerably increase the computation time for the verification of liveness or bounded-response properties. Indeed, even with few pieces, the memory needed to perform classic verification of such properties is too large for the verification to be run to completion.

6 Related works

Several works have been devoted to study behavioral equivalences, as well as their associated preorder, in the timed case.

Concerning equivalences, timed bisimulation was studied in (Cerans 1992). Three time-abstracting bisimulations were introduced in (Tripakis & Yovine 2001): strong time-abstracting bisimulation, observational time-abstracting bisimulation and delay time-abstracting bisimulation. Contrary to the timed bisimulation, time-abstracting bisimulations abstract away from quantitative aspects of time elapsing. Strong time-abstracting bisimulation preserves reachability, timed Büchi automata and CTL (TCTL is also preserved modulo a transformation of the TA and of the formula as presented in (Tripakis 1998)). Observational and delay time-abstracting bisimulations preserve reachability and timed Büchi automata. Therefore, bisimulations preserve a wide range of properties. However, they are not really adapted to incremental modeling.

Preorders, to which we are interested in this paper, are more adapted and have also been widely studied. Time-abstracting simulation equivalence and preorder have been studied in (Henzinger, Henzinger & Kopke 1995), but timed properties are not preserved by these relations. Timed simulation was defined in (Tasiran, Alur, Kurshan & Brayton 1996). The authors showed that the problem of checking if a TA simulates another one w.r.t. the relation they defined is solvable in EXPTIME. The relation has the properties of composability, compatibility and compositionality w.r.t. a totally synchronous composition operator. However, non-observable actions are not considered in the definition of this simulation.

The notion of simulation which seems the closest to the one we defined in this paper is the timed ready simulation of (Jensen, Larsen & Skou 2000). It is defined on extended timed automata, i.e., timed automata which can contain urgent actions and shared variables. Non-observable actions are also considered. The definition of this relation is almost equivalent to the definition of our timed τ-simulation, and thus preserves safety properties. But, it has not been extended to handle the preservation of liveness properties. It has the properties of composability, compatibility and compositionality w.r.t. a composition operator which paradigm is close to the one of the classic parallel composition we considered. However, an assumption concerning the absence of internal activity (i.e., the absence of τ-transitions) in the automata is done to benefit of these properties, in particular, for the compositionality property.

To our knowledge, there is no simulation preorder which has been defined in the timed case, taking into account non-observable actions, and preserving in particular liveness properties.

7 Conclusion and future works

The context of this paper is the verification by model-checking of component-based timed systems, modeled by timed automata. To cope with the state-space explosion problem of model-checking, we propose to develop such systems incrementally, either by refinement or by integration of components. These methods allow to check properties on smaller-sized models, where model-checking is still applicable. The main issue for these methods to be applicable concerns the preservation of properties, established on these smaller-sized models, on the complete model.

To guarantee the preservation, we defined τ-simulation relations for timed systems, with preservation abilities: a timed τ-simulation which ensures the preservation of safety properties, and a divergence-sensitive and stability-respecting one which handles

the preservation of all MITL properties, as well as strong non-zenoness and deadlock-freedom. We have shown that the timed τ-simulation is well-adapted to incremental development with the classic parallel composition operator for timed automata. Indeed, it is compatible with the operator, and composability and compositionality hold. This is not the case for the DS timed τ-simulation, due to the fact that deadlocks often appear when composition is achieved with this operator. Composability is the fact that a component simulates its integration with other ones. Its direct consequence is thus that properties are automatically preserved during integration of components. As this property does not hold in the case of the classic operator for the DS timed τ-simulation, it is essential to check that the algorithmic verification of the preservation, by means of the simulation, does not remove interest to incremental development. We implemented the verification of the DS timed τ-simulation in a tool named VESTA, in the particular case of integration of components. We made experiments to check if, even if the simulation is checked algorithmically, the cost in practice of an incremental verification by integrating components is still lower than the cost of a direct verification on the complete model. It turns out that, on a production cell example and in terms of computation times, the methodology we propose appears to be more efficient.

This case study (as well as the study of the CSMA/CD protocol that we presented in (Bellegarde, Julliand, Mountassir & Oudot 2006*a*)) shows the interest of the method for some kind of parameterized systems, when a system S is likely to contain an undetermined number of *identical* components $C_i, i = 1..n$, modulo some renaming. The number n of these components can be viewed as a parameter of the system. For instance, in the production cell example, the number of pieces which are admitted in the cell is this parameter. In the case of integration of components with the classic operator, it would be interesting to be able to check properties with a small fixed number m of components, such that preservation is ensured whatever the number n of components, with $n \geq m$. For this purpose, it seems sufficient to get conditions on the $C_i's$ and/or on their synchronizations to guarantee that adding more of these components in the system does not introduce deadlocks (in addition to the condition on the absence of non-zeno cycles of internal activity in the components). Thus, an interesting work is to study such conditions.
Another work direction concerns the problem of implementability of timed automata. It consists in studying if the properties which are established on a timed automaton are preserved on the implementation corresponding to it, on a given platform. The main issue comes from the fact that the semantics of timed automata is often considered as being ideal: execution times of actions are ignored, clocks are infinitely precise, etc... In practice, these *perfect* concepts are not true. To check properties on the implementation, (Altisen & Tripakis 2005) proposes an approach based on modeling, to benefit of the existing verification tools for timed automata. They propose to model the execution platform P as timed automata, and to transform the timed automaton A modeling the system into an untimed finite automaton. The composition of all these automata, called the execution model M, represents the execution of A on the platform P. An interesting perspective would be to study the compatibility of the simulations we defined in the specific framework of the timed automaton A and its execution model M, while taking into account the way the platform is specified. This could also allow to exhibit conditions on the modeling of the platform (and thus on the *real* platform) which guarantee the preservation of the properties of A on its implementation on the platform.

References and Notes

Altisen, K. & Tripakis, S. (2005), Implementation of Timed Automata : an Issue of Semantics or Modeling ?, Technical Report TR-2005-12, Verimag, Grenoble, France.

Alur, R. (1991), Techniques for Automatic Verification of Real-Time Systems, PhD thesis, Department of Computer Science, Stanford University.

Alur, R., Courcoubetis, C. & Dill, D. (1993), 'Model-Checking in Dense Real-time', *Information and Computation* **104**(1), 2–34.

Alur, R. & Dill, D. (1994), 'A theory of timed automata', *Theoretical Computer Science* **126**(2), 183–235.

Alur, R., Feder, T. & Henzinger, T. (1996), 'The benefits of relaxing punctuality', *Journal of the ACM* **43**, 116–146.

Bellegarde, F., Julliand, J. & Kouchnarenko, O. (2000), Ready-simulation is not Ready to Express a Modular Refinement Relation, *in* 'Proceedings of the 3^{rd} International Conference on Fondamental Aspects of Software Engineering (FASE'00)', Vol. 1783 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 266–283.

Bellegarde, F., Julliand, J., Mountassir, H. & Oudot, E. (2005), On the contribution of a τ-simulation in the incremental modeling of timed systems, *in* 'Proceedings of the 2^{nd} International Workshop on Formal Aspects of Component Software (FACS'05)', Vol. 160 of *Electronic Notes in Theoretical Computer Science*, Elsevier, Macao, Macao, pp. 97–111.

Bellegarde, F., Julliand, J., Mountassir, H. & Oudot, E. (2006*a*), Experiments in the use of τ-simulations for the components-verification of real-time systems, *in* 'Proceedings of the 5^{th} International Workshop on

Specification And Verification of Component-Based Systems (SAVCBS'06)', Portland, Oregon, USA. Also available on ACM Digital Library.

Bellegarde, F., Julliand, J., Mountassir, H. & Oudot, E. (2006*b*), The Tool VeSTA: Verification of Simulations for Timed Automata, Technical Report RT2006-01, LIFC, Laboratoire d'Informatique de l'Université de Franche-Comté.

Burns, A. (2003), 'How to verify a safe real-time system: The application of model-checking and timed automata to the production cell case study', *Real-Time Systems Journal* **24**(2), 135–152.

Cerans, K. (1992), Decidability of bisimulation equivalence for parallel timer processes, *in* 'Proceedings of the 4^{th} workshop on Computer-Aided Verification (CAV'92)', Vol. 663 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 302–315.

Clarke, E. & Emerson, E. (1981), Design and synthesis of synchronization skeletons using branching-time temporal logic, *in* 'Proceedings of Workshop on Logic of Programs', Vol. 131 of *Lecture Notes in Computer Science*, Springer-Verlag.

Emerson, E. & Halpern, J. (1982), Decision procedures and expressiveness in the temporal logic of branching time, *in* 'Proceedings of the 14^{th} ACM Symp. Theory of Computing (STOC'82)', San Francisco, CA, USA, pp. 169–180.

Glabbeck, R. v. (1990), The Linear Time - Branching Time Spectrum, *in* 'Proceedings of the 1^{st} international Conference on Concurrency Theory (CONCUR'90)', Vol. 458 of *Lecture Notes in Computer Science*, Springer-Verlag, Amsterdam, Netherlands, pp. 278–297.

Glabbeck, R. v. (1993), The Linear Time - Branching Time Spectrum II ; The semantics of sequential systems with silent moves, *in* 'Proceedings of 4^{th} international Conference on Concurrency Theory (CONCUR'93)', Vol. 715 of *Lecture Notes in Computer Science*, Springer-Verlag, Hildesheim, Germany, pp. 66–81.

Henzinger, M., Henzinger, T. & Kopke, P. (1995), Computing simulations on finite and infinite graphs, *in* 'Proceedings of the 36^{th} IEEE Symposium on Foundations of Computer Science', pp. 453–462.

Hoare, C. (1985), *Communicating Sequential Processes*, Prentice Hall.

Jensen, H., Larsen, K. & Skou, A. (2000), Scaling up Uppaal : Automatic verification of real-time systems using compositionnality and abstraction, *in* 'Proceedings of the 6^{th} international symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00)', Springer-Verlag, London, UK, pp. 19–30.

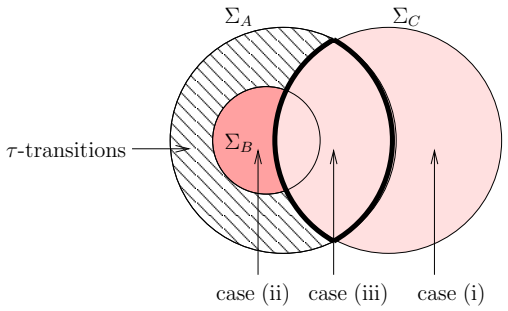

Figure 11 Different types of transitions for the proof of Proposition 4

Pnueli, A. (1981), 'The temporal semantics of concurrent programs', *Theoretical Computer Science* **13**, 1–20.

Tasiran, S., Alur, R., Kurshan, R. & Brayton, R. (1996), Verifying Abstractions of Timed Systems, *in* 'Proceedings of the 7^{th} Conference on Concurrency Theory (CONCUR'96)', Vol. 1119 of *Lecture Notes in Computer Science*, Pisa, Italy, pp. 546–562.

Tripakis, S. (1998), The analysis of timed systems in practice, PhD thesis, Universite Joseph Fourier, Grenoble, France.

Tripakis, S. & Yovine, S. (2001), 'Analysis of Timed Systems using Time-abstracting Bisimulations', *Formal Methods in System Design* **18**(1), 25–68.

Tripakis, S., Yovine, S. & Bouajjani, A. (2005), 'Checking Timed Büchi Automata Emptiness Efficiently', *Formal Methods in System Design* **26**(3), 267–292.

Yovine, S. (1997), 'Kronos: A verification tool for real-time systems', *Journal of Software Tools for Technology Transfer* **1**(1/2), 123–133.

Appendix A. Proof of Proposition 4

Consider a relation $R \subseteq S_{A\|C} \times S_{B\|C}$ such that $(s_A, s_C)R(s_B, s'_C)$ if $s_A \mathcal{S} s_B$ and $s_C = s'_C$. As previously, we prove that R satisfy clauses 1 to 3 of Definition 4. Let $((s_A, s_C), (s_B, s_C)) \in R$,

1. *Strict simulation* : the proof is naturally divided into three parts. Indeed, this clause concerns transitions labelled in $\Sigma_A \cup \Sigma_C$, divided in three types of transitions in $A\|C$, comparing to $B\|C$, as shown in Fig. 11:

   (i) Transitions of C which do not synchronize with an action of A (this case corresponds to the right side of Σ_C in Fig. 11),

(ii) Transitions in Σ_B[8] which do not synchronize with an action of C (the part of Σ_B in dark grey in Fig. 11),

(iii) Transitions in C which synchronize with a transition in A. They appear in $B\|C$ either as interleaving actions of C if the synchronization is done with an action of A which does not appear in B, or as an action of B synchronized with an action of C otherwise (the part in the center inside the bold line in Fig. 11).

Let us detail these three cases:

(i) Consider a transition $(s_A, s_C) \stackrel{g,c,r}{\rightarrow} (s_A, s'_C)$ such that $c \in \Sigma_C \backslash \Sigma_A$. By construction of $A\|C$, a transition $s_C \stackrel{g,c,r}{\rightarrow} s'_C$ exists in C. Therefore, g only involves clocks of C and $v_C \in g$. Thus, by construction of $B\|C$, a transition $(s_B, s_C) \stackrel{g,c,r}{\rightarrow} (s_B, s'_C)$ exists in $B\|C$. Since $s_A \mathcal{S} s_B$, and by definition of R, we have $(s_A, s'_C) R (s_B, s'_C)$.

(ii) Consider a transition $(s_A, s_C) \stackrel{g,a,r}{\rightarrow} (s'_A, s_C)$ such that $a \in (\Sigma_A \cap \Sigma_B) \backslash \Sigma_C$ (the state s_C is not modified). By definition of $\|$, the transition $s_A \stackrel{g,a,r}{\rightarrow} s'_A$ exists in A. Since $s_A \mathcal{S} s_B$, there is a transition $s_B \stackrel{g',a,r'}{\rightarrow} s'_B$ in B, such that $s'_A \mathcal{S} s'_B$. Thus, $v_B \in g'$. Since g' only involves clocks of B and that the set of clocks of B and C are disjoint (by hypothesis in the construction of $B\|C$), then $(v_B, v_C) \in g'$ and the transition $(s_B, s_C) \stackrel{g',a,r'}{\rightarrow} (s'_B, s_C)$ exists in $B\|C$ and $(s'_A, s_C) R (s'_B, s_C)$ by definition of R.

(iii) Consider a transition $(s_A, s_C) \stackrel{g,a,r}{\rightarrow} (s'_A, s'_C)$ such that $a \in \Sigma_A \cap \Sigma_C$. By definition of $\|$, there is a transition $s_A \stackrel{g_1,a,r_1}{\rightarrow} s'_A$ in A and a transition $s_C \stackrel{g_2,a,r_2}{\rightarrow} s'_C$ in C. There are two cases: either $a \in \Sigma_A \cap \Sigma_B$ (a is an observable action of A comparing to B and thus exists in B), or $a \in \Sigma_A \backslash \Sigma_B$ (a is a non-observable action of A which do not exists in B).
In the first case, since $s_A \mathcal{S} s_B$, there is a transition $s_B \stackrel{g_3,a,r_3}{\rightarrow} s'_B$ in B such that $s'_A \mathcal{S} s'_B$. Thus, there is a transition $(s_B, s_C) \stackrel{g',a,r'}{\rightarrow} (s'_B, s'_C)$ in $B\|C$, such that $(s'_A, s'_C) R (s'_B, s'_C)$ by definition of R.
In the second case, since $s_A \mathcal{S} s_B$, we have $s'_A \mathcal{S} s_B$. The transition $(s_B, s_C) \stackrel{g_2,a,r_2}{\rightarrow} (s_B, s'_C)$ exists in $B\|C$ since v_B do not involve clocks of C and $v_C \in g_2$. By definition of R, we have $(s'_A, s'_C) R (s_B, s'_C)$.

Thus, the relation R satisfies the *strict simulation*.

2. *Delay equality*: consider a time transition $(s_A, s_C) \stackrel{t}{\rightarrow} (s'_A, s'_C)$. By definition of $\|$, the transitions $s_A \stackrel{t}{\rightarrow} s'_A$ and $s_C \stackrel{t}{\rightarrow} s'_C$ exist respectively in A and C. Since $s_A \mathcal{S} s_B$, then the transition $s_B \stackrel{t}{\rightarrow} s'_B$ exists in B and $s'_A \mathcal{S} s'_B$. The transition $(s_B, s_C) \stackrel{t}{\rightarrow} (s'_B, s'_C)$ exists also in $A\|C$ and, by definition of R, $(s'_A, s'_C) R (s'_B, s'_C)$.

3. *τ-transitions stuttering*: in $A\|C$, comparing to $B\|C$, τ-transitions are labelled in $\Sigma_A \backslash (\Sigma_B \cup \Sigma_C)$ (the hatched part in Fig. 11). Consider a transition $(s_A, s_C) \stackrel{\tau}{\rightarrow} (s'_A, s_C)$ (the state s_C is not modified since τ represents an action of $\Sigma_A \backslash (\Sigma_B \cup \Sigma_C)$). Since $s_A \mathcal{S} s_B$, we have $s'_A \mathcal{S} s_B$. It follows that $(s'_A, s_C) R (s_B, s_C)$.

4. *Location labelling respect*: immediate since $s_A \mathcal{S} s_B$.

5. *Common clocks valuation equality*: immediate since $s_A \mathcal{S} s_B$.

Appendix B. Proof of Lemma 4

The proof is done by fixed-point induction. Consider the function $\mathcal{F} : Z_2 \times Z_1 \rightarrow Z_2 \times Z_1$ defined in the following way. If $\mathcal{Z}_{ds} \subseteq Z_2 \times Z_1$, then $((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds})$ iff the clauses 1 to 6 of Definition 6 hold.
In the definition of $\preceq_{\mathcal{Z}_{ds}}$, we consider the greatest fixed-point of the function $\mathcal{F}$. This function is trivially monotonic (and the sets Z_2 and Z_1 are finite), thus we can reason inductively about the pre-fixed-points of this function. The principle of this induction is the following: given a predicate P, this induction allows to prove $P(\mathcal{Z}_{ds})$, by proving $P(\mathcal{F}(\mathcal{Z}_{ds}))$, with the induction assumption that $P(\mathcal{Z}_{ds})$ holds.
Therefore, we suppose that $(\star)$ is true for $\mathcal{Z}_{ds}$, and prove that $(\star)$ holds for $\mathcal{F}(\mathcal{Z}_{ds})$. In other words, we prove that

$$((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds}) \Rightarrow \forall v_2 \cdot (v_2 \in \zeta_2 \Rightarrow \exists v_1 \cdot (v_1 \in \zeta_1 \wedge (q_2, v_2) \mathcal{S}_{ds} (q_1, v_1))).$$

We proceed clause by clause. For clarity's sake, we specify exactly the formula we intend to prove for each clause. We begin with the clause *common clocks valuation equality* since we use it in the rest of the proof.

1. *Common clocks valuations equality*: we prove that

$$((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds}) \Rightarrow \forall v_2 \cdot (v_2 \in \zeta_2 \Rightarrow \exists v_1 \cdot (v_1 \in \zeta_1 \wedge v_2 \lfloor_{X_1} = v_1)).$$

By definition of the function $\mathcal{F}$, if $((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds})$, then $\zeta_2 \lfloor_{X_1} \subseteq \zeta_1$. The proof is immediate by definition of the operator $\lfloor$.

2. *Strict simulation*: we prove
$((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds}) \Rightarrow \forall v_2 \cdot (v_2 \in \zeta_2 \wedge (q_2, v_2) \stackrel{e_2}{\rightarrow} (q'_2, v'_2) \wedge \texttt{label}(e_2) \in \Sigma_1 \Rightarrow$

$$\exists v_1, v'_1, q'_1 \cdot (v_1 \in \zeta_1 \wedge (q_1, v_1) \stackrel{e_1}{\rightarrow} (q'_1, v'_1) \wedge \texttt{label}(e_1) = \texttt{label}(e_2) \wedge (q'_2, v'_2) \mathcal{S}_{ds} (q'_1, v'_1))).$$

Consider a transition $(q_2, v_2) \xrightarrow{e_2} (q_2', v_2')$ such that $\texttt{label}(e_2) \in \Sigma_1$. By construction of the simulation graph, if such a transition exists, then a transition $(q_2, \zeta_2) \xrightarrow{e_2} (q_2', \zeta_2')$ exists in $SG(A, c)$, such that $v_2 \in \zeta_2$ and $v_2' \in \zeta_2'$. Since $((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds})$, there exists a transition $(q_1, \zeta_1) \xrightarrow{e_1} (q_1', \zeta_1')$ such that $\texttt{label}(e_1) = \texttt{label}(e_2)$ which satisfies the clause *strict simulation* of the definition of $\mathcal{F}$. In particular, we have $(q_2', \zeta_2')\mathcal{Z}_{ds}(q_1', \zeta_1')$. Since this transition exists, and by construction of the simulation graph, there exists at least one discrete (semantic) transition inscribed in this transition. Moreover, we prove that there exists $v_1 \in \zeta_1$ such that $v_2\rfloor_{X_1} = v_1$. Since $\texttt{src\_val}((q_2, \zeta_2), e_2, (q_2', \zeta_2'))\rfloor_{X_1} \subseteq \texttt{src\_val}((q_1, \zeta_1), e_1, (q_1', \zeta_1'))$, there exists a discrete transition $(q_1, v_1) \xrightarrow{e_1} (q_1', v_1')$ such that $v_1' \in \zeta_1'$. It remains to prove that $(q_2', v_2')\mathcal{S}_{ds}(q_1', v_1')$. The action which labels e_2 is an observable action of A_2. Thus, we know that the transition resets exactly the same clocks in X_1 than e_1, i.e., $\texttt{reset}(e_2) \cap X_1 = \texttt{reset}(e_1)$. Moreover, since $v_2\rfloor_{X_1} = v_1$, and that, by definition

$$v_2' = [\texttt{reset}(e_2) := 0]v_2 \text{ and } v_1' = [\texttt{reset}(e_1) := 0]v_1$$

we have $v_2'\rfloor_{X_1} = v_1'$. Since $(q_2', \zeta_2')\mathcal{Z}_{ds}(q_1', \zeta_1')$, and $(\star)$ holds for $\mathcal{Z}_{ds}$, we have $(q_2', v_2')\mathcal{S}_{ds}(q_1', v_1')$.

3. *Delay equality*: we prove that
$((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds}) \Rightarrow$
$\forall v_2 \cdot (v_2 \in \zeta_2 \wedge (q_2, v_2) \xrightarrow{t} (q_2, v_2') \Rightarrow \exists v_1, v_1' \cdot (v_1 \in \zeta_1 \wedge (q_1, v_1) \xrightarrow{t} (q_1, v_1') \wedge (q_2, v_2')\mathcal{S}_{ds}(q_1, v_1')))$.

   Time transitions which appear in a semantic transition system do not appear explicitly as transitions in the associated simulation graph. Time elapses intuitively inside zones. Consider a transition $(q_2, v_2) \xrightarrow{t} (q_2, v_2')$ inside the zone (q_2, ζ_2) (i.e., $v_2 \in \zeta_2$ and $v_2' \in \zeta_2$). Consider also the valuation $v_1 \in \zeta_1$ such that $v_2\rfloor_{X_1} = v_1$ (this valuation exists by the clause *Common clocks valuations equality*). Since $\zeta_2\rfloor_{X_1} \subseteq \zeta_1$, there exists a transition $(q_1, v_1) \xrightarrow{t} (q_1, v_1')$ such that $v_1' \in \zeta_1$ and $v_1' = v_2'\rfloor_{X_1}$. With the induction assumption, we know that there exists a valuation v_1'' such that $(q_2, v_2')\mathcal{S}_{ds}(q_1, v_1'')$ and $v_1'' = v_2'\rfloor_{X_1}$. Thus, $v_1'' = v_1'$. It follows that $(q_2, v_2')\mathcal{S}_{ds}(q_1, v_1')$.

4. *τ-transitions stuttering*: we prove that
$((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds}) \Rightarrow$

$$\forall v_2 \cdot (v_2 \in \zeta_2 \wedge (q_2, v_2) \xrightarrow{e_2} (q_2', v_2') \wedge \texttt{label}(e_2) = \tau \Rightarrow \exists v_1 \cdot (v_1 \in \zeta_1 \wedge (q_2', v_2')\mathcal{S}_{ds}(q_1, v_1))).$$

   Consider a transition $(q_2, v_2) \xrightarrow{e_2} (q_2', v_2')$ such that $\texttt{label}(e_2) = \tau$. By construction of the simulation graph, a transition $(q_2, \zeta_2) \xrightarrow{e_2} (q_2', \zeta_2')$ exists in $SG(A, c)$, such that $v_2 \in \zeta_2$ and $v_2' \in \zeta_2'$. Since $((q_2, \zeta_2), (q_1, \zeta_1)) \in \mathcal{F}(\mathcal{Z}_{ds})$, and by definition of $\mathcal{F}$, we have $(q_2', \zeta_2')\mathcal{Z}_{ds}(q_1, \zeta_1)$. We proved previously that there exists a valuation $v_1 \in \zeta_1$ such that $v_1 = v_2\rfloor_{X_1}$. It remains to prove that $v_2'\rfloor_{X_1} = v_1$. Since τ-transitions only reset clocks in $X_2 \backslash X_1$, only the valuations of the clocks in X_2 are changed in v_2' in comparison to v_2. Thus, $v_2'\rfloor_{X_1} = v_1$. Since $(q_2', \zeta_2')\mathcal{Z}_{ds}(q_1, \zeta_1)$, and $(\star)$ holds for $\mathcal{Z}_{ds}$, we have $(q_2', v_2')\mathcal{S}_{ds}(q_1, v_1)$.

5. *Location labelling respect*: immediate by the definition of $\mathcal{F}$.

6. *Divergence-sensitivity*: immediate by lemma 3.

7. *Stability-respect*: we prove that

$$\text{if } (\zeta_2 \backslash \texttt{free}(q_2))\rfloor_{X_1} \subseteq \zeta_1 \backslash \texttt{free}(q_1) \text{ then } \forall v_2 \cdot (v_2 \in \zeta_2 \wedge v_2 \notin \texttt{free}(q_2) \Rightarrow \exists v_1 \cdot (v_1 \in \zeta_1 \wedge v_1 \notin \texttt{free}(q_1))).$$

   This is equivalent to:
   if $\forall v \cdot (v \in (\zeta_2 \backslash \texttt{free}(q_2))\rfloor_{X_1} \Rightarrow \exists v' \cdot (v' \in \zeta_1 \backslash \texttt{free}(q_1) \wedge v' = v))$

   then $\forall v_2 \cdot (v_2 \notin \texttt{free}(q_2) \Rightarrow \exists v_1 \cdot (v_1 \notin \texttt{free}(q_1) \wedge v_2\rfloor_{X_1} = v_1))$.

   Therefore, we have immediately that $(\star)$ holds $\mathcal{F}(\mathcal{Z}_{ds})$.

Notes

[1] Safety properties express that *on some conditions, something never happens.*

[2] Liveness properties express that *on some conditions, something will eventually happen.*

[3] The work presented in this section (except sections 3.3.2 and 3.3.3) was first presented in (Bellegarde, Julliand, Mountassir & Oudot 2005).

[4] Note that we do not intend to directly check algorithmically this semantic definition. It will be extended into a symbolic relation (see section 5) where this clause *delay equality* consists in polyhedra inclusion and thus, is decidable.

[5] In the simulation graph, only discrete transitions appear. Thus, a τ-cycle is a cycle which only contains transitions labelled by τ.

[6] VeSTA at: `http://lifc.univ-fcomte.fr/~oudot/VeSTA`

[7] This case study, as well as other experimentations, was first presented in (Bellegarde et al. 2006*a*).

[8] These transitions are also in Σ_A, since $A \preceq_S B$ and therefore $\Sigma_B \subseteq \Sigma_A$.